Αλγόριθμοι και Αποτελέσματα NP-πληρότητας για Προβλήματα Χρωματισμού και Επικάλυψης με Μονοπάτια σε Τέλεια Γραφήματα

Η ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης

του Τμήματος Πληροφορικής Εξεταστική Επιτροπή

από την

Αικατερίνη Ασδρέ

ως μέρος των Υποχρεώσεων για τη λήψη του

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ

ΠΛΗΡΟΦΟΡΙΚΗ

Ιούνιος 2008

Ευχαριστιες

Απευθύνω τις θερμότερες ευχαριστίες μου στον επιβλέποντα καθηγητή μου κ. Σταύρο Δ. Νικολόπουλο για την πολύτιμη βοήθεια και ουσιαστική συμβολή του στην επιτυχή περάτωση της διδακτορικής μου διατριβής. Η συμβολή του ήταν καθοριστική χάρη στην επιστημονική του γνώση και βοήθειά του στην αυστηρή κατασκευή του "γραφήματος" της έρευνάς μου και υπόδειξη των βέλτιστων "διαδρομών" του σε "πολυωνυμικό" χρόνο. Θα ήθελα να του εκφράσω την ευγνωμοσύνη μου, αφού χωρίς την συνεχή καθοδήγηση και την αμέριστη υποστήριξή του, αυτή η εργασία δεν θα μπορούσε να ολοκληρωθεί. Επιπλέον, τον ευχαριστώ θερμά για την ενθουσιώδη και συνεχή προτροπή του για συνέχιση της έρευνας των "όμορφων" διαδρομών του πεδίου της αλγοριθμικής θεωρίας γραφημάτων.

Θα ήθελα να ευχαριστήσω τον καθηγητή του Τμήματος Πληροφορικής κ. Λεωνίδα Παληό (μέλος της τριμελούς συμβουλευτικής επιτροπής) για την βοήθεια, τις συμβουλές και τον χρόνο που μου αφιέρωσε. Οι συμβουλές του ήταν καθοριστικές σε όλα τα ερευνητικά θέματα που μελετήσαμε. Επίσης, θα ήθελα να ευχαριστήσω τον καθηγητή κ. Βασίλειο Ζησιμόπουλο (μέλος της τριμελούς συμβουλευτικής επιτροπής) για την επιστημονική συμβολή του στην εκπόνηση της διατριβής μου. Τέλος, ευχαριστώ θερμά τους κκ. Ευστάθιο Ζάχο, Αντώνιο Συμβώνη, Ιωάννη Μανωλόπουλο, και Χρήστο Νομικό (μέλη της επταμελούς εξεταστικής επιτροπής) για τις εύστοχες και εποικοδομητικές παρατηρήσεις τους σε αρκετά από τα θέματα της μελέτης μου.

Θα ήταν παράλειψη να μην αναφερθώ στην αμέριστη βοήθεια, στη συνεχή υποστήριξη και στην υψηλού επιπέδου υλικοτεχνική υποδομή που μου παρείχε το Τμήμα Πληροφορικής καθ' όλη τη διάρκεια των μεταπτυχιακών μου σπουδών. Επιπλέον, θεωρώ εξαιρετικά σημαντικό το γεγονός ότι είχα τη δυνατότητα να εργάζομαι σε ένα φιλικό και ευχάριστο περιβάλλον, όπως αυτό του Τμήματος Πληροφορικής. Ειλικρινά, είμαι ευγνώμων.

Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για τη συμπαράσταση και την ηθική υποστήριξή τους κατά τη διάρκεια των μεταπτυχιακών μου σπουδών.

Αικατερίνη Ασδρέ

Ιωάννινα, 27 Ιουνίου 2008

Περιληψη

Αυτή η εργασία επικεντρώνεται στη μελέτη της πολυπλοκότητας προβλημάτων επικάλυψης με μονοπάτια και χρωματισμού σε κλάσεις τέλειων γραφημάτων. Συγκεκριμένα, ασχολούμαστε με την σχεδίαση και ανάλυση αλγορίθμων για αυτά τα προβλήματα ή, για τις περιπτώσεις που αυτό δεν είναι δυνατό, δίνουμε αποτελέσματα NP-πληρότητας. Τα περισσότερα αποτελέσματα αυτής της εργασίας έχουν ήδη δημοσιευτεί. Στη συνέχεια περιγράφουμε επεκτάσεις προβλημάτων επικάλυψης και χρωματισμού που μας απασχολούν στη συγκεκριμένη εργασία.

Παράλληλοι αλγόριθμοι για το πρόβλημα της επικάλυψης με μονοπάτια
Επικεντρώνουμε το ενδιαφέρον μας στην ανάπτυξη ενός βέλτιστου παράλληλου αλγορίθμου που υπολογίζει τον μικρότερο αριθμό των ξένων (ως προς τους κόμβους) μεταξύ τους μονοπατιών που καλύπτουν τους κόμβους (γνωστό ως το πρόβλημα επικάλυψης με μονοπάτια–path cover problem) ενός P_4-sparse γραφήματος. Επωφελούμαστε της δομής του md-δέντρου (και στην πραγματικότητα των πρώτων (prime) γραφημάτων του) και εκμεταλλευόμαστε τα μονοπάτια δέντρων (path trees–μια δομή που διατηρεί τα μονοπάτια της επικάλυψης μονοπατιών). Πιο συγκεκριμένα, δοθέντος του md-δέντρου ενός P_4-sparse γραφήματος G σε n κόμβους, χρησιμοποιούμε κλασικές δενδρικές τεχνικές συρρίκνωσης και ταίριασμα παρενθέσεων (bracket matching) και περιγράφουμε ένα βέλτιστο παράλληλο αλγόριθμο που τρέχει σε $O(\log n)$ χρόνο με $O(n/\log n)$ επεξεργαστές σε EREW-PRAM μοντέλο. Είναι ενδιαφέρον κανείς να επεκτείνει αυτήν την τεχνική σε άλλες κλάσεις γραφημάτων, και ακόμα περισσότερο, να θεωρήσει μερικές επεκτάσεις του προβλήματος, όπως είναι το πρόβλημα επικάλυψης με μονοπάτια καθορισμένων άκρων (k-fixed-endpoint path cover), σε P_4-sparse ή σε άλλες κλάσεις γραφημάτων.

Αυτή η εργασία οδήγησε στην δημοσίευση:

- K. Asdre, S.D. Nikolopoulos and C. Papadopoulos, An optimal solution for the path cover problem on P_4-sparse graphs, *Workshop on Graphs and Combinatorial Optimization (CTW'05)*, Cologne, Germany, 2005.
- K. Asdre, S.D. Nikolopoulos and Ch. Papadopoulos, An optimal parallel solution for the path cover problem on P_4-sparse graphs, *Journal of Parallel and Distributed Computing* **67**, 63–76, 2007.

Το πρόβλημα της επικάλυψης με μονοπάτια μήκους το πολύ k (k-path partition problem)
Μελετούμε το πρόβλημα της επικάλυψης με μονοπάτια μήκους το πολύ k και, με κίνητρο μια πρόσφατη εργασία του Steiner [87], όπου άφησε ανοιχτό το πρόβλημα για την κλάση των convex γραφημάτων, αποδεικνύουμε ότι το πρόβλημα είναι NP-πλήρες για τη συγκεκριμένη κλάση. Επίσης αποδεικνύουμε ότι το πρόβλημα της επικάλυψης με μονοπάτια μήκους το πολύ k είναι NP-πλήρες για την κλάση των quasi-threshold γραφημάτων.

Αυτή η μελέτη συμπεριλαμβάνεται στην παρακάτω εργασία:

- K. Asdre and S.D. Nikolopoulos, NP-completeness results for some problems on subclasses of bipartite and chordal graphs, *Theoret. Comput. Science* **381**, 248–259, 2007.

Εύρεση Αρμονικού και Αχρωματικού Αριθμού
Επεκτείνοντας προηγούμενα αποτελέσματα NP-πληρότητας για τα προβλήματα εύρεσης του αρμονικού και του αχρωματικού αριθμού σε δέντρα (trees), διμερή γραφήματα (bipartite graphs) και συμπληρωματικά παραγόμενα γραφήματα (cographs), αποδεικνύουμε ότι τα προβλήματα αυτά είναι επίσης NP-πλήρη για συνδεδεμένα γραφήματα διαστημάτων (interval graphs), συνδεδεμένα μεταθετικά γραφήματα (permutation graphs) καθώς και για διμερή μεταθετικά γραφήματα (bipartite permutation graphs). Επιπλέον, μελετούμε την πολυπλοκότητα αυτών των ποβλημάτων σε δύο γνωστές υποκλάσεις των τριγωνικών (chordal) γραφημάτων, τα quasi-threshold και τα threshold γραφήματα. Με βάση μια εργασία του Bodlaender [11], παρουσιάζουμε αποτελέσματα NP-πληρότητας για τα προβλήματα εύρεσης του αρμονικού και του αχρωματικού αριθμού σε quasi-threshold γραφήματα. Επίσης δείχνουμε ότι το πρόβλημα εύρεσης του αρμονικού αριθμού είναι NP-πλήρες για τα διαχωρίσιμα (split) γραφήματα. Ακόμη, είναι γνωστό ότι το πρόβλημα λύνεται πολυωνυμικά σε threshold γραφήματα. Δείχνουμε ότι και το πρόβλημα εύρεσης του αχρωματικού αριθμού λύνεται πολυωνυμικά στη συγκεκριμένη κλάση γραφημάτων, περιγράφοντας ένα γραμμικό αλγόριθμο.

Αυτή η εργασία οδήγησε στις εξής δημοσιεύσεις:

- K. Asdre and S.D. Nikolopoulos, NP-completeness results for some problems on subclasses of bipartite and chordal graphs, *Theoret. Comput. Science* **381**, 248 - 259, 2007.
- K. Asdre, K. Ioannidou, and S.D. Nikolopoulos, The harmonious coloring problem is NP-complete for interval and permutation graphs, *Discrete Appl. Math.* **155**, 2377–2382, 2007.

Το πρόβλημα της επικάλυψης με μονοπάτια με k καθορισμένα άκρα (k-fixed-endpoint path cover problem)
Μελετούμε μια επέκταση του προβλήματος επικάλυψης με μονοπάτια που ονομάζουμε πρόβλημα της επικάλυψης με μονοπάτια με k καθορισμένα άκρα (k-fixed-endpoint path cover problem), ή για συντομία, kPC. Δοθέντος ενός γραφήματος G και ενός υποσυνόλου $\mathcal{T}$ που περιέχει k κόμβους του $V(G)$, μια επικάλυψη με μονοπάτια με k καθορισμένα άκρα του G είναι ένα σύνολο $\mathcal{P}$ από ξένα (ως προς τους κόμβους) μεταξύ τους μονοπάτια που καλύπτει τους κόμβους του G τέτοια ώστε οι k κόμβοι του $\mathcal{T}$ να είναι άκρα των μονοπατιών του $\mathcal{P}$. Το πρόβλημα της επικάλυψης με μονοπάτια με k καθορισμένα άκρα είναι η εύρεση μιας επικάλυψης με μονοπάτια με k καθορισμένα άκρα ελάχιστης πληθικότητας. Σημειώνουμε ότι, αν το $\mathcal{T}$ είναι κενό, δηλαδή αν $k = 0$, το πρόβλημα ταυτίζεται με το κλασσικό πρόβλημα επικάλυψης με μονοπάτια. Δείχνουμε ότι το πρόβλημα της επικάλυψης με μονοπάτια με k καθορισμένα άκρα λύνεται σε πολυωνυμικό χρόνο για την κλάση των cographs. Συγκεκριμένα, υπολογίζουμε ένα κάτω φράγμα για το μέγεθος μιας επικάλυψης με μονοπάτια με k καθορισμένα άκρα ελάχιστης πληθικότητας για ένα cograph και αποδεικνύουμε ιδιότητες των μονοπατιών μιας τέτοιας επικάλυψης. Στη συνέχεια, με βάση αυτές τις ιδιότητες, περιγράφουμε έναν αλγόριθμο ο οποίος υπολογίζει σε γραμμικό χρόνο ($O(n+m)$) μια επικάλυψη με μονοπάτια με k καθορισμένα άκρα ελάχιστης πληθικότητας σε ένα cograph G με n κόμβους και m ακμές. Ο προτεινόμενος αλγόριθμος είναι απλός, απαιτεί γραμμικό χώρο και μας επιτρέπει να λύσουμε και άλλα σχετικά προβλήματα, οπως το 1HP και το 2HP στην κλάση των cographs με την ίδια πολυπλοκότητα χρόνου και χώρου.

Το 1993, ο Damaschke παρατήρησε ότι η πολυπλοκότητα των προβλημάτων 1HP και 2HP problems για την κλάση των γραφημάτων διατημάτων (interval graphs) παραμένει ένα ανοιχτό πρόβλημα [27]. Λόγω αυτής της παρατήρησης δείχνουμε ότι το kPC πρόβλημα μπορεί να λυθεί σε γραμμικό χρόνο ($O(n+m)$) για την κλάση των proper interval γραφημάτων. Μελετούμε επίσης το πρόβλημα της επικάλυψης με μονοπάτια με ένα καθορισμένο άκρο (για συντομία, 1PC) σε interval graphs, το οποίο αποτελεί ειδική περίπτωση του προβλήματος kPC και γενίκευση του προβλήματος 1HP. Έτσι, δείχνουμε ότι το πρόβλημα 1PC μπορεί να λυθεί σε πολυωνυμικό χρόνο για την κλάση των interval γραφημάτων. Ο προτεινόμενος αλγοριθμος απαιτεί $O(n^2)$ χρόνο, γραμμικό χώρο και μας επιτρέπει να λύσουμε το πρόβλημα 1HP σε interval γραφήματα με την ίδια πολυπλοκότητα χρόνου και χώρου.

Αυτή η εργασία οδήγησε στις εξής δημοσιεύσεις:

- K. Asdre and S.D. Nikolopoulos, A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs, *Networks* **50**, 231–240, 2007.
- K. Asdre and S.D. Nikolopoulos, A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs, *Proc. 18th International Conference on Combinatorial Algorithms (IWOCA'07)*, Lake Macquarie, Newcastle, Australia, 2007.
- K. Asdre and S.D. Nikolopoulos, The 1-fixed-endpoint path cover problem is polynomial on interval graphs, submitted to Algorithmica.
- K. Asdre and S.D. Nikolopoulos, A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs, submitted to Theoret. Comput. Science.

Το πρόβλημα της επικάλυψης με μονοπάτια με k σύνολα καθορισμένων άκρων (k-fixed-endpoint-set Path Cover Problem)

Μελετούμε μια επέκταση του προβλήματος επικάλυψης με μονοπάτια που ονομάζουμε πρόβλημα της επικάλυψης με μονοπάτια με k σύνολα καθορισμένων άκρων (k-fixed-endpoint-set path cover problem), ή για συντομία, kFSPC. Δοθέντος ενός γραφήματος G και k ξένων μεταξύ τους υποσυνόλων, $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ του $V(G)$, μια επικάλυψη με μονοπάτια με k σύνολα καθορισμένων άκρων του G είναι ένα σύνολο $\mathcal{P}$ από ξένα (ως προς τους κόμβους) μεταξύ τους μονοπάτια που καλύπτει τους κόμβους του G τέτοια ώστε οι κόμβοι του $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ να είναι άκρα των μονοπατιών του $\mathcal{P}$ και δύο κόμβοι $u, v \in \mathcal{T}$ ανήκουν στο ίδιο μονοπάτι του $\mathcal{P}$ αν οι u, v ανήκουν στο ίδιο σύνολο $\mathcal{T}^i$, $i \in [1, k]$. Το πρόβλημα της επικάλυψης με μονοπάτια με k σύνολα καθορισμένων άκρων είναι η εύρεση μιας επικάλυψης με μονοπάτια με k σύνολα καθορισμένων άκρων ελάχιστης πληθικότητας. Σημειώνουμε ότι, αν το $\mathcal{T}^i$, $\forall i \in [1, k]$ είναι κενό το πρόβλημα ταυτίζεται με το κλασσικό πρόβλημα επικάλυψης με μονοπάτια και επίσης, το πρόβλημα της επικάλυψης με μονοπάτια με ένα σύνολο καθορισμένων άκρων ταυτίζεται με το πρόβλημα kPC [?]. Έτσι, το πρόβλημα kFSPC γενικεύει το πρόβλημα kPC καθώς και τα προβλήματα 1HP και 2HP, τα οποία είναι NP-πλήρη σε γενικά γραφήματα. Δείχνουμε ότι το πρόβλημα kFSPC λύνεται σε πολυωνυμικό χρόνο για την κλάση των cographs. Ο προτεινόμενος γραμμικός αλγόριθμος είναι απλός και απαιτεί γραμμικό χώρο.

Το πρόβλημα της επικάλυψης με μονοπάτια με 2 τερματικά σύνολα (2-terminal-set Path Cover Problem)

Μελετούμε το πρόβλημα της επικάλυψης με μονοπάτια με 2 τερματικά σύνολα (2-terminal-set path cover problem), ή για συντομία, 2TPC, που αποτελεί επέκταση του προβλήματος επικάλυψης με μονοπάτια. Δοθέντος ενός γραφήματος G και 2 ξένων μεταξύ τους υποσυνόλων, $\mathcal{T}^1$ και $\mathcal{T}^2$ του $V(G)$, μια επικάλυψη με μονοπάτια με 2 τερματικά σύνολα του G είναι ένα σύνολο $\mathcal{P}$ από ξένα (ως προς τους κόμβους) μεταξύ τους μονοπάτια που καλύπτει τους κόμβους του G τέτοια ώστε οι κόμβοι του $\mathcal{T}^1$ και $\mathcal{T}^2$ να είναι άκρα των μονοπατιών του $\mathcal{P}$ και όλα τα μονοπάτια που έχουν και τα δύο άκρα τους στο σύνολο $\mathcal{T}^1 \cup \mathcal{T}^2$, έχουν το ένα άκρο στο $\mathcal{T}^1$ και το άλλο στο $\mathcal{T}^2$. Το πρόβλημα της επικάλυψης με μονοπάτια με 2 τερματικά σύνολα είναι η εύρεση μιας επικάλυψης με μονοπάτια με 2 τερματικά σύνολα ελάχιστης πληθικότητας. Σημειώνουμε ότι, αν το $\mathcal{T}^1 \cup \mathcal{T}^2$ είναι κενό το πρόβλημα ταυτίζεται με το κλασσικό πρόβλημα επικάλυψης με μονοπάτια.

Επίσης, το πρόβλημα 2TPC γενικεύει τα προβλήματα 1HP και 2HP. Δείχνουμε ότι το πρόβλημα 2TPC λύνεται σε πολυωνυμικό χρόνο για την κλάση των cographs. Ο προτεινόμενος γραμμικός αλγόριθμος είναι απλός, απαιτεί γραμμικό χώρο και μας δίνει τη δυνατότητα να λύσουμε το 1HP και το 2HP στα cographs με την ίδια πολυπλοκότητα χρόνου και χώρου.

Αυτή η εργασία οδήγησε στις εξής δημοσιεύσεις:

- K. Asdre and S.D. Nikolopoulos, The 2-terminal-set path cover problem and its polynomial solution on cographs, *Frontiers of Algorithmics Workshop (FAW '08)*, Changsha, China, 2008.
- K. Asdre and S.D. Nikolopoulos, The 2-terminal-set path cover problem and its polynomial solution on cographs, submitted to Discrete Appl. Math.

Το πρόβλημα της επικάλυψης με μονοπάτια με k τερματικά σύνολα πληθικότητας το πολύ 2 ($k(2)$-terminal-set Path Cover Problem)

Η πολυπλοκότητα του προβλήματος της επικάλυψης με μονοπάτια με k τερματικά σύνολα πληθικότητας το πολύ 2 ($k(2)$-terminal-set path cover problem), ή για συντομία k(2)TSPC, εξετάζεται επίσης για την κλάση των cographs. Δοθέντος ενός γραφήματος G και k ξένων μεταξύ τους υποσυνόλων $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ του $V(G)$ που το κάθε ένα περιέχει το πολύ δύο κόμβους, μια επικάλυψη με μονοπάτια με τερματικά σύνολα πληθικότητας το πολύ 2 του G είναι ένα σύνολο $\mathcal{P}$ από ξένα (ως προς τους κόμβους) μεταξύ τους μονοπάτια που καλύπτει τους κόμβους του G τέτοια ώστε οι κόμβοι του $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ να είναι άκρα των μονοπατιών του $\mathcal{P}$ και όλα τα μονοπάτια που έχουν και τα δύο άκρα τους στο σύνολο $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ έχουν το ένα άκρο στο $\mathcal{T}^i$ και το άλλο στο $\mathcal{T}^j$, $i \neq j$ και $i, j \in [1, k]$. Το πρόβλημα k(2)TSPC είναι η εύρεση μιας επικάλυψης με μονοπάτια με k τερματικά σύνολα πληθικότητας το πολύ 2 με ελάχιστη πληθικότητα. Σημειώνουμε ότι, αν το $\mathcal{T}^i$, $\forall i \in [1, k]$ είναι κενό το πρόβλημα ταυτίζεται με το κλασσικό πρόβλημα επικάλυψης με μονοπάτια, ενώ αν $|\mathcal{T}^i| = 1$, $\forall i \in [1, k]$ το πρόβλημα ταυτίζεται με το πρόβλημα kPC [?]. Επίσης, το πρόβλημα k(2)TSPC γενικεύει τα προβλήματα 1HP και 2HP. Δείχνουμε ότι το πρόβλημα k(2)TSPC λύνεται σε πολυωνυμικό χρόνο για την κλάση των cographs. Τέλος, θα ήταν ενδιαφέρον να μελετήσει κανείς την πολυπλοκότητα του προβλήματος επικάλυψης με μονοπάτια με k τερματικά σύνολα (για συντομία kTSPC) για την κλάση των cographs. Το πρόβλημα kTSPC αποτελεί γενίκευση του k(2)TSPC. Η διαφορά των δύο προβλημάτων είναι ότι στο πρώτο δεν υπάρχει περιορισμός στο πλήθος των κόμβων που περιέχουν τα σύνολα $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$.

Extended Abstract in English

This work focuses on the complexity status of path cover and coloring problems on classes of perfect graphs. In particular, it deals with designing and analyzing graph algorithms for these problems or, for the cases where this is not possible, with providing NP-completeness results. Most of our results have already been published. We next describe the variations of the path cover and coloring problems that we study in this work.

Parallel algorithms for the path cover problem
We focus on developing an optimal parallel algorithm to find and report the smallest number of vertex-disjoint paths that cover the vertices (known as the path cover problem) for the class of P_4-sparse graphs. We take advantage of the structure of its modular decomposition tree (and in fact its prime graphs) and utilize its path trees (a structure that maintains the paths of the path cover). Specifically, given the modular decomposition tree of a P_4-sparse graph G on n vertices, we use standard tree contraction and bracket matching techniques, and we describe an optimal parallel algorithm which runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW-PRAM model. It would be interesting to extend this technique to other classes of graphs and, furthermore, to use this technique in order to solve some variants of the problem, such as the k-fixed-endpoint path cover, on P_4-sparse or on other classes of graphs.

This work lead to the following publications:

- K. Asdre, S.D. Nikolopoulos and C. Papadopoulos, An optimal solution for the path cover problem on P_4-sparse graphs, *Workshop on Graphs and Combinatorial Optimization (CTW'05)*, Cologne, Germany, 2005.
- K. Asdre, S.D. Nikolopoulos and Ch. Papadopoulos, An optimal parallel solution for the path cover problem on P_4-sparse graphs, *Journal of Parallel and Distributed Computing* **67**, 63–76, 2007.

The k-path partition problem
We study the k-path partition problem and, motivated by a recent work of Steiner [87], where he left the problem open for the class of convex graphs, we prove that the k-path partition problem is NP-complete on convex graphs. We also prove that the problem is NP-complete on quasi-threshold graphs.

This work is published in:

- K. Asdre and S.D. Nikolopoulos, NP-completeness results for some problems on subclasses of bipartite and chordal graphs, *Theoret. Comput. Science* **381**, 248–259, 2007.

Harmonious and Pair-Complete Coloring
Extending previous NP-completeness results for the harmonious coloring problem and the pair-complete coloring problem on trees, bipartite graphs and cographs, we prove that these problems are also NP-complete on connected interval and permutation graphs and also on bipartite permutation graphs. Moreover, we study the complexity of these problems on two well-known subclasses of chordal graphs, namely quasi-threshold and threshold graphs. Based on the work of Bodlaender [11], we show NP-completeness results for the pair-complete coloring and harmonious coloring problems on quasi-threshold graphs. We also show that the harmonious coloring problem is NP-complete on split graphs. It is known that the harmonious coloring problem is polynomially solvable on threshold graphs. We show that the pair-complete coloring problem is also polynomially solvable on threshold graphs by describing a linear-time algorithm.

This work lead to the following publications:

- K. Asdre and S.D. Nikolopoulos, NP-completeness results for some problems on subclasses of bipartite and chordal graphs, *Theoret. Comput. Science* **381**, 248 - 259, 2007.
- K. Asdre, K. Ioannidou, and S.D. Nikolopoulos, The harmonious coloring problem is NP-complete for interval and permutation graphs, *Discrete Appl. Math.* **155**, 2377–2382, 2007.

The k-fixed-endpoint path cover problem
We study a variant of the path cover problem, namely, the k-fixed-endpoint path cover problem, or kPC, for short. Given a graph G and a subset $\mathcal{T}$ of k vertices of $V(G)$, a k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the k vertices of $\mathcal{T}$ are all endpoints of the paths in $\mathcal{P}$. The k-fixed-endpoint path cover problem is to find a k-fixed-endpoint path cover of G of minimum cardinality; note that, if $\mathcal{T}$ is empty, that is, $k = 0$, the stated problem coincides with the classical path cover problem. We show that the k-fixed-endpoint path cover problem can be solved in linear time on the class of cographs. More precisely, we first establish a lower bound on the size of a minimum k-fixed-endpoint path cover of a cograph and prove structural properties for the paths of such a path cover. Then, based on these properties, we describe an algorithm which, for a cograph G on n vertices and m edges, computes a minimum k-fixed-endpoint path cover of G in linear time, that is, in $O(n+m)$ time. The proposed algorithm is simple, requires linear space, and also enables us to solve some path cover related problems, such as the 1HP and 2HP, on cographs within the same time and space complexity.

In 1993, Damaschke stated that the complexity status of both 1HP and 2HP problems on interval graphs remains an open question [27]. Motivated by this, we show that the kPC problem can be solved in linear time, that is, in $O(n + m)$ time, on the class of proper interval graphs. We also study the 1-fixed-endpoint path cover problem on interval graphs, or 1PC for short, which is a special case of the kPC problem and a generalization of the 1HP problem. We show that the 1PC problem can be solved in polynomial time on the class of interval graphs. The proposed algorithm runs in $O(n^2)$ time, requires linear space, and enables us to solve the 1HP problem on interval graphs within the same time and space complexity.

This study lead to the following publications:

- K. Asdre and S.D. Nikolopoulos, A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs, *Networks* **50**, 231–240, 2007.
- K. Asdre and S.D. Nikolopoulos, A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs, *Proc. 18th International Conference on Combinatorial Algorithms (IWOCA'07)*, Lake Macquarie, Newcastle, Australia, 2007.

- K. Asdre and S.D. Nikolopoulos, The 1-fixed-endpoint path cover problem is polynomial on interval graphs, submitted to Algorithmica.
- K. Asdre and S.D. Nikolopoulos, A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs, submitted to Theoret. Comput. Science.

The k-fixed-endpoint-set Path Cover Problem

We study a generalization of the path cover problem, namely, the k-fixed-endpoint-set path cover problem, or kFSPC for short. Given a graph G and k disjoint subsets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of $V(G)$, a k-fixed-endpoint-set path cover of G is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are all endpoints of the paths in $\mathcal{P}$ and two vertices $u, v \in \mathcal{T}$ belong to the same path of $\mathcal{P}$ if both u, v belong to the same set $\mathcal{T}^i$, $i \in [1, k]$; a *minimum k-fixed-endpoint-set path cover* of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a k-fixed-endpoint-set path cover of G with minimum cardinality; the *k-fixed-endpoint-set path cover problem* (kFSPC) is to find a minimum k-fixed-endpoint-set path cover of the graph G. Note that, if $\mathcal{T}^i$, $\forall i \in [1, k]$ is empty, the stated problem coincides with the classical path cover problem, while the 1-fixed-endpoint-set path cover problem coincides with the k-fixed-endpoint path cover problem (kPC) [?]. Thus, the kFSPC problem generalizes the kPC problem and also the 1HP and 2HP problems, which have been proved to be NP-complete in general graphs. We show that the kFSPC problem can be solved in linear time on the class of cographs. The proposed linear-time algorithm is simple and requires linear space.

The 2-terminal-set Path Cover Problem

We study the 2-terminal-set path cover problem, or 2TPC for short, which is a generalization of the path cover problem. Given a graph G and two disjoint subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of $V(G)$, a 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ are all endpoints of the paths in $\mathcal{P}$ and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$. The 2TPC problem is to find a 2-terminal-set path cover of G of minimum cardinality; note that, if $\mathcal{T}^1 \cup \mathcal{T}^2$ is empty, the stated problem coincides with the classical path cover problem. The 2TPC problem also generalizes the 1HP and 2HP problems. We show that the 2TPC problem can be solved in linear time on the class of cographs. The proposed linear-time algorithm is simple, requires linear space, and also enables us to solve the 1HP and 2HP problems on cographs within the same time and space complexity.

This work lead to the following publications:

- K. Asdre and S.D. Nikolopoulos, The 2-terminal-set path cover problem and its polynomial solution on cographs, *Frontiers of Algorithmics Workshop (FAW '08)*, Changsha, China, 2008.
- K. Asdre and S.D. Nikolopoulos, The 2-terminal-set path cover problem and its polynomial solution on cographs, submitted to Discrete Appl. Math.

The $k(2)$-terminal-set Path Cover Problem

The $k(2)$-terminal-set path cover problem, or k(2)TSPC for short, is also studied for the class of cographs. Given a graph G and k disjoint subsets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of $V(G)$ each containing at most two vertices, a $k(2)$-terminal-set path cover of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ are all endpoints of the paths in $\mathcal{P}$ and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ have one endpoint in $\mathcal{T}^i$ and the other in $\mathcal{T}^j$, $i \neq j$ and $i, j \in [1, k]$. The k(2)TSPC problem is to find a $k(2)$-terminal-set path cover of G of minimum cardinality; note that, if $\mathcal{T}^i$, $\forall i \in [1, k]$ is empty, the stated problem coincides with the classical path cover problem, while if $|\mathcal{T}^i| = 1$, $\forall i \in [1, k]$ the problem coincides with the k-fixed-endpoint path cover problem [**?**]. The k(2)TSPC problem also generalizes the 1HP and 2HP problems. We show that the k(2)TSPC problem can be solved in linear time on the class of cographs. It would be interesting to study the complexity of the k-terminal-set Path Cover Problem on cographs, or kTSPC for short, which is a generalization of the k(2)TSPC problem. The difference between the two problems is that for the kTSPC problem there is no restriction to the size of the sets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Preliminaries

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively. A subgraph H of a graph G is said to be induced if, for any pair of vertices x and y of H, xy is an edge of H if and only if xy is an edge of G. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V(G) - S]$ and by $G - v$ the graph $G[V(G) - \{v\}]$. The complement $\overline{G}$ of a graph G has the same vertex set as G, and distinct vertices u, v are adjacent in $\overline{G}$ if and only if they are not adjacent in G.

The *neighborhood* $N_G(x)$ of a vertex x of the graph G is the set of all the vertices of G which are adjacent to x. The *closed neighborhood* of x is defined as $N_G[x] := N_G(x) \cup \{x\}$. The set of vertices in $V(G) - \{x\}$ that are not neighbors of x is its *non-neighbors* and is denoted by $\overline{N}_G(x)$. When it is clear which the graph we refer to is, we use the notation $N(x)$ and $N[x]$ for simplicity. The *degree* of a vertex x in a graph G, denoted by $d(x)$, is the number of edges incident on x; thus, $d(x) = |N(x)|$. A *clique* is a set of pairwise adjacent vertices while a *stable* (or *independent*) *set* is a set of pairwise non-adjacent vertices.

A sequence of vertices $[v_0, v_1, \dots, v_k]$ is a *walk* from v_0 to v_k of length k in G provided that $v_{i-1}v_i \in E(G)$ for $i = 1, 2, \dots, k$. A walk in G is called a *path* if no vertex and no edge occurs more than once. A path of length k is called *trivial* if $k = 0$. A sequence of vertices $[v_0, v_1, \dots, v_k, v_0]$ is called a *cycle* of length $k + 1$ if $v_{i-1}v_i \in E(G)$ for $i = 1, 2, \dots, k$ and $v_kv_0 \in E(G)$. A cycle $[v_0, v_1, \dots, v_k, v_0]$ is a *simple cycle* if $v_i \neq v_j$ for $i \neq j$. A simple cycle $[v_0, v_1, \dots, v_k, v_0]$ is *chordless* if $v_iv_j \notin E(G)$ for every non-successive vertices v_i, v_j in the cycle. Throughout this work, a clique on n vertices is denoted by K_n, a chordless cycle on n vertices is denoted by C_n and the chordless path on n vertices is denoted by P_n.

In a chordless path $P_n = [v_1, v_2, \ldots, v_{n-1}, v_n]$ the vertices $v_2, \ldots, v_{n-1}$ are called *midpoints* or *internal vertices* and v_1 and v_n are called *endpoints.*

Let T be a graph with n vertices. Then, T is a tree if and only if it contains no cycles and has $n-1$ edges; equivalently, T is a tree if and only if any two vertices of T are connected by exactly one path.

Let T be a tree. The parent of a node x of T is denoted by $p(x)$ and the set of nodes containing the children of node x in T is denoted by $ch(x)$. Furthermore, we denote by L_i the set of nodes belonging to level i in T for $i = 0$ to $i = h$, where h is the height of the tree T.

1.2 Perfect Graphs

A *hole* of G is an induced subgraph of G which is a cycle of length at least 4. An *antihole* of G is an induced subgraph of G whose complement is a hole in G. A graph G is *Berge* if every hole and antihole of G has even length.

The *chromatic number* $\chi(G)$ of a graph G is the fewest number of colors needed to properly color the vertices of G, or equivalently, the fewest number of stable sets needed to cover the vertices of G. The *clique number* $\omega(G)$ of a graph G is the size of the largest complete subgraph of G. Clearly, the chromatic number of a graph G is greater than or equal to the clique number of G. The *stability number* $\alpha(G)$ of a graph G is the size of the largest stable set of G while the *clique cover number* $\kappa(G)$ of a graph G is the fewest number of complete subgraphs needed to cover the vertices of G.

The intersection of a clique and a stable set of a graph G can be at most one vertex. Thus, for any graph G,

$$\omega(G) \leq \chi(G) \text{ and } \alpha(G) \leq \kappa(G).$$

These equalities are dual to one another since $\alpha(G) = \omega(\overline{G})$ and $\kappa(G) = \chi(\overline{G})$. Let $G(V, E)$ be an undirected graph. The following three conditions are the *perfection properties* of a graph G.

(P1) $\omega(G[A]) = \chi(G[A]), \quad \forall A \subseteq V(G)$

(P2) $\alpha(G[A]) = \kappa(G[A]), \quad \forall A \subseteq V(G)$

(P3) $\omega(G[A]) \cdot \alpha(G[A]) \geq |A|, \quad \forall A \subseteq V(G)$

Theorem 1.1 (The Perfect Graph Theorem [7]). *For an undirected graph G the perfection properties P1, P2 and P3 are equivalent.*

A graph G is *perfect* if for every induced subgraph $G[A]$ of G, the chromatic number of $G[A]$ equals the size of the largest clique of $G[A]$. The study of perfect graphs was initiated by Claude Berge, partly motivated by a problem from information theory (finding the "Shannon capacity" of a graph; it lies between the size of the largest clique and the chromatic number, and so for a perfect graph it equals both). In particular, in 1961 Berge [6] proposed two celebrated conjectures about perfect graphs. Since the second implies the first, they were known as the "weak" and "strong" perfect graph conjectures respectively, although both are now theorems, the following:

Theorem 1.2. *The complement of every perfect graph is perfect.*

Theorem 1.3. *A graph is perfect if and only if it is Berge.*

The first was proved by Lovász [65] in 1972. The second, the *strong perfect graph conjecture*, received a great deal of attention over the past 40 years. In May 2002, Maria Chudnovsky and Paul Seymour announced that they, building on earlier joint work with Neil Robertson and Robin Thomas, had completed the proof of the Strong Perfect Graph Conjecture which became the *strong perfect graph theorem*. The four joint authors published the 178-page paper in 2006 [16]. As a result, holes and antiholes have been extensively studied in many different contexts in algorithmic graph theory. Thus, finding a hole or an antihole in a graph efficiently is an important graph-theoretic problem, both on its own and as a step in many recognition algorithms. In 2004, Nikolopoulos and Palios [74] proposed a $O(n+m^2)$ time algorithm for the problem of finding a hole or an antihole in a graph on n vertices and m edges, requiring $O(nm)$ space.

The class of perfect graphs has structural properties leading to polynomial time algorithms for optimization problems which are NP-complete in arbitrary graphs, such as coloring and path cover problems and the problems of finding Hamilton cycle or path. These problems find applications in many fields of different sciences, from mathematics to philosophy [7, 41].

1.3 Complexity and Intractability

Besides providing a basis for comparing algorithms which solve the same problem, algorithmic analysis has other practical uses. Most importantly, it affords us the opportunity to know in advance an estimate of the computation or a bound on the storage and run time requirements. Let us refer to the differences between computability and computational complexity [40]. Computability addresses itself mostly to questions of existence: Is there an algorithm which solves problem Π? Proving that a problem is computable usually consists of demonstrating an actual algorithm which will terminate with a correct answer for every input. The amount of resources (time and space) used in the calculation, although finite, is unlimited.

In contrast to this, computational complexity deals precisely with the quantitative aspects of problem solving. It addresses the issue of what can be computed within a practical or reasonable amount of time and space by measuring the resource requirements exactly or by obtaining upper and lower bounds. Complexity is actually determined on three levels: the problem, the algorithm, and the implementation. Naturally, we want the best algorithm which solves our problem, and we want to choose the best implementation of that algorithm.

A *decision problem* is one which requires a simple "yes" or "no" answer. An *instance* of a problem Π is a specification of particular values for its parameters. Usually we can rewrite an optimization problem as a decision problem which at first seems much easier to solve than the original but turns out to be just about as hard. Consider the following two versions of the graph coloring problem.

Graph Coloring (optimization version)
Instance: An undirected graph G.
Question: What is the smallest number of colors needed for a proper coloring of G?

Graph Coloring (decision version)
Instance: An undirected graph G and an integer $k > 0$.
Question: Does there exist a proper k coloring of G?

The optimization version can be solved by applying an algorithm for the decision version n times for an n-vertex graph. If the n decision problems are solved sequentially, then the time needed to solve the optimization version is larger than that for the decision version by at most a factor of n. However, if they can be solved in parallel, then the time needed for both versions is essentially the same. Demonstrating and analyzing the complexity of a particular algorithm for Π provides us with an upper bound on the complexity of Π. Determining the complexity of a problem Π requires a two-sided attack:

(1) the upper bound –the minimum complexity over all known algorithms solving Π.

(2) the lower bound–the largest function f for which it has been proved mathematically that all possible algorithms solving Π are required to have complexity at least as high as f.

Our ultimate goal is to make these bounds coincide. A gap between (1) and (2) tells us how much more research is needed to achieve this goal. The biggest open question involving the gap between upper and lower complexity bounds involves the so called NP-complete problems. For each of the problems in this class only exponential-time algorithms are known, yet the best lower bounds proved so far are polynomial functions. Furthermore, if a polynomial time algorithm exists for one of them, then such an algorithm exists for all of them. Included among the NP-complete problems on graphs are finding a Hamiltonian circuit, a minimum coloring, or a maximum clique.

1.3.1 Polynomial Transformations and NP-completeness

The *state* of an algorithm consists of the current values of all variables and the location of the current instruction to be executed. A *deterministic algorithm* is one for which each state upon execution of the instruction uniquely determines at most one next state. Virtually all computers run deterministically. A problem Π is in the class P if there exists a deterministic polynomial time algorithm which solves Π.

A *nondeterministic algorithm* is one for which a state may determine many next states and which follows up on each of the next states simultaneously. We may regard a nondeterministic algorithm as having the capability of branching off into many copies of itself, one for each next state. Thus, while a deterministic algorithm must explore a set of alternatives one at a time, a nondeterministic algorithm examines all alternatives at the same time.

A problem Π is in the class NP if there exists a nondeterministic polynomial time algorithm which solves Π. Clearly, P $\subseteq$ NP. An important open question in the theory of computation is whether the containment of P in NP is proper; i.e., is P $\neq$ NP?

One problem Π_1 is *polynomially transformable* to another problem Π_2 denoted $\Pi_1 \preccurlyeq \Pi_2$, if there exists a function f mapping the instances of Π_1 into the instances of Π_2 such that

(i) f is computable deterministically in polynomial time, and

(ii) a solution to the instance $f(I)$ of Π_2, gives a solution to the instance I of Π_1, for all I.

Intuitively this means that Π_1 is no harder to solve than Π_2 up to added polynomial term, for we could solve Π_1 by combining the transformation f with the best algorithm for solving Π_2. Thus, if $\Pi_1 \preccurlyeq \Pi_2$, then

$$\text{COMPLEXITY}(\Pi_1) \leq \text{COMPLEXITY}(\Pi_2) + \text{POLYNOMIAL}.$$

If Π_2 has a deterministic polynomial time algorithm, then so does Π_1; if every deterministic algorithm solving Π_1 requires at least an exponential amount of time, then the same is true for Π_2.

The principal technique used for demonstrating that two problems are related is that of "reducing" one to the other, by giving a constructive transformation that maps any instance of the first problem into an equivalent instance of the second. Such a transformation provides the means for converting any algorithm that solves the second problem into a corresponding algorithm for solving the first problem.

A problem Π is *NP-hard* if any one of the following equivalent conditions holds:

(1) $\Pi' \preccurlyeq \Pi$ for all $\Pi' \in$ NP;

(2) $\Pi \in$ P $\Rightarrow$ P=NP;

(3) the existence of a deterministic polynomial time algorithm for Π would imply the existence of a polynomial time algorithm for every problem in NP.

A problem Π is *NP-complete* if it is both a member of NP and it is NP-hard. The NP-complete problems are the most difficult of those in NP. To prove that Π is NP-complete we show that $\Pi \in$ NP and some known NP-complete problem Π' transforms to Π.

The foundations for the theory of NP-completeness were laid in a paper of Stephen Cook [19], presented in 1971, entitled "The complexity of theorem proving procedures". He emphasized the significance of "polynomial time reducibility", that is, reductions for which the required transformation can be executed by a polynomial time algorithm. If we have a polynomial time reduction from one problem to another, this ensures that any polynomial time algorithm for the second problem can be converted into a corresponding polynomial time algorithm for the first problem. Furthermore, he focused attention on the class of NP of decision problems that can be solved in polynomial time by a nondeterministic computer, and he proved that one particular problem in NP, called the "satisfiability" problem, has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability can be solved with a polynomial time algorithm, then so can every problem in NP, and if any problem in NP is intractable (i.e. it is so hard that no polynomial time algorithm can possibly solve it), then the satisfiability problem also must be intractable. Thus, in a sense, the satisfiability problem is the "hardest" problem in NP.

Finally, Cook suggested that other problems in NP might share with the satisfiability this property of being the "hardest" member of NP. He showed this to be the case for the problem "Does a given graph G contain a complete subgraph on a given number k of vertices?"

Subsequently, Karp [56] presented a collection of results proving that indeed the decision versions of many well known combinatorial problems, including the travelling salesman problem, are just as "hard" as the satisfiability problem. Since then a wide variety of other problems have been proved equivalent in difficulty to these problems, and this equivalence class, consisting of the "hardest" problems in NP, has been given a name: the class of NP-complete problems; in their book [36], "A guide to the theory of NP-completeness", Garey and Johnson provide a list of NP-complete problems. However, a graph theoretic or other type of problem Π which is normally hard to solve in the general case may have an efficient solution if the input domain is suitably restricted. The Hamiltonian circuit problem, for example, is trivial if the only graphs considered are trees.

1.3.2 Number Problems and Strong NP-completeness

We refer to the PARTITION problem, which is NP-complete [36]:

PARTITION
Instance: A finite set A and a "size" $s(a) \in Z^+$ for each $a \in A$.
Question: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

Consider now the following "dynamic programming" approach to solving the PARTITION problem. Let $A = \{a_1, a_2, \ldots, a_n\}$ and let $T(i, j)$, $1 \leq i \leq n$, $0 \leq j \leq b$, denote the truth value of the statement "there exists a subset of $\{a_1, a_2, \ldots, a_i\}$ whose sum is exactly j". To start with, we know that $T(1, j)$ is false for all j except $j = 0$ and $j = a_1$. Recursively, we have that $T(i + 1, j)$, $1 \leq i < n$, $0 \leq j \leq b$, is true if and only if either $T(i, j)$ is true or $a_{i+1} \leq j$ and $T(i, j - a_{i+1})$ is true. The desired subset for PARTITION exists if and only if $T(n, b)$ is true.

The obvious iterative algorithm based on this formulation can solve the PARTITION problem in $O(nb)$ time. Although this may appear to be a polynomial time algorithm, in fact it is not. For the purposes of the theory of NP-completeness, as well as for most formal complexity theory, the time complexity of an algorithm is expressed in terms of a single "instance size" parameter which reflects the number of symbols that would be required to describe the instance in a "reasonable" and "concise" manner. The parameter nb in the example does not do this. If an integer N is to be described "concisely", it must be represented using only $O(logN)$ symbols, as would be the case using its binary or decimal representations. Thus, an instance of PARTITION could be described with only $O(nlogb)$ symbols, and nb is not bounded by any polynomial function of this. For this reason, our $O(nb)$ algorithm for PARTITION is not a polynomial time algorithm; we shall refer to it as a "pseudo-polynomial" time algorithm. However, the existence of such an algorithm does indicate that the NP-completeness of PARTITION depends strongly on the fact that arbitrarily large values of the a_i are allowed. If any upper bound were imposed on these numbers in advance, even a bound which is a polynomial function of n, this algorithm would be a polynomial time algorithm for the restricted problem. Thus, one must take care when interpreting an NP-completeness result for a problem involving numbers not to infer intractability in a broader sense than is justified by the theory. The ordinary NP-completeness results for such problems leave open the possibility of algorithms, which are, for certain applications, suitably efficient to be used in practice.

However, not all NP-complete problems are like PARTITION. We say that a problem Π is a *number problem* if there exists no polynomial p such that $Max[I] \leq p(Length[I])$ for all instances I of Π [36]. Note that, the function $Length$ is intended to map any instance I to an integer $Length[I]$ that corresponds to a number of symbols used to describe I under some reasonable encoding scheme for Π. The function Max is intended to map any instance I to an integer $Max[I]$ that corresponds to the magnitude of the largest number in I. Furthermore, an algorithm that solves a problem Π will be called a ***pseudo-polynomial time algorithm*** for Π if its time complexity function is bounded above by a polynomial function of the two variables $Length[I]$ and $Max[I]$ [35, 36]. Note that, even though an NP-completeness result for a problem Π rules out the possibility of solving Π with a polynomial time algorithm (unless P=NP), it does not rule out the possibility of solving Π with a pseudo-polynomial time algorithm.

As an immediate consequence of the above definition of a number problem, we can make the following observation.

Observation 1.1. [36] If Π is NP-complete and Π is not a number problem, then Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

Thus, assuming that P$\neq$NP, the only NP-complete problems that are potential candidates for being solved by pseudo-polynomial time algorithms are those that are number problems.

For any decision problem Π and any polynomial p (over the integers), let Π_p denote the subproblem of Π obtained by restricting Π to only those instances I that satisfy $Max[I] \leq p(Length[I])$. Then Π_p is not a number problem. Furthermore, if Π is solvable by a pseudo-polynomial time algorithm, then Π_p must be solvable by a polynomial time algorithm. Given any input string x, all we need to do is check that x encodes an instance I satisfying $Max[I] \leq p(Length[I])$ and, if so, apply the pseudo-polynomial time algorithm for Π to I. The required inequality can be checked in polynomial time. By the definition of pseudo-polynomial time algorithm, the algorithm for Π will be a polynomial time algorithm for the instances that satisfy this inequality. Thus, a decision problem Π is ***NP-complete in the strong sense*** if Π belongs to NP and there exists a polynomial p over the integers for which Π_p is NP-complete. In particular, if Π is NP-complete and Π is not a number problem, then Π is automatically NP-complete in the strong sense. We then have the following generalization of Observation 1.1.

Observation 1.2. [36] If Π is NP-complete in the strong sense, then Π cannot be solved by a pseudo-polynomial time algorithm unless P=NP.

The second observation provides the means for applying the theory of NP-completeness to questions about the existence of pseudo-polynomial time algorithms. We know that PARTITION cannot be NP-complete in the strong sense while TRAVELLING SALESMAN, 3-PARTITION, SEQUENCING

WITHIN INTERVALS and SUBFOREST ISOMORPHISM are problems which are NP-complete in the strong sense.

1.4 Path Cover and Coloring Problems

We next describe the problems this work is concerned with, that is, path cover and coloring problems. We also give the motivation of our work, mostly by describing the large amount of related work until today.

1.4.1 Path Cover Problem and Variations

A well studied problem with numerous practical applications in graph theory is to find a minimum number of vertex-disjoint paths of a graph G that cover the vertices of G. This problem, also known as the path cover problem (PC), finds application in the fields of database design, networks, code optimization among many others (see [2, 4, 61, 86]); it is well known that the path cover problem and many of its variants are NP-complete in general graphs [36]. It is known to be NP-complete even when the inputs are restricted to several interesting special classes of graphs, such as bipartite graphs [3], edge graphs [12], chordal graphs [20] and several interesting classes of intersection graphs [9, 71]. A graph that admits a path cover of size one is referred to as Hamiltonian. Thus, the path cover problem is at least as hard as the Hamiltonian path problem (HP), that is, the problem of deciding whether a graph is Hamiltonian.

Several variants of the HP problem are also of great interest, among which is the problem of deciding whether a graph admits a Hamiltonian path between two points (2HP). The 2HP problem is the same as the HP problem except that in 2HP two vertices of the input graph G are specified, say, u and v, and we are asked whether G contains a Hamiltonian path beginning with u and ending with v. Similarly, the 1HP problem is to determine whether a graph G admits a Hamiltonian path starting from a specific vertex u of G, and to find one if such a path does exist. Both 1HP and 2HP problems are also NP-complete in general graphs [36].

The path cover problem and several variants of it have numerous algorithmic applications in many fields. Some that have received both theoretical and practical attention are in the content of communication and/or transposition networks [89]. In such problems, we are given a graph (network) G and

(Problem A) a set $\mathcal{T}$ of $k = 2\lambda$ vertices of G, and the objective is to determine whether G admits a path cover of size λ that contains paths connecting pairs of vertices of $\mathcal{T}$, that is, G admits λ vertex-disjoint paths with both their endpoints in $\mathcal{T}$ (note that, the endpoints of a path P are the first vertex and the last vertex visited by P), or

(Problem B) a set $\mathcal{T}$ of $\lambda = k/2$ pairs of vertices of G (source-sink pairs), and the objective is to determine whether G admits for each pair (a_i, b_i), $1 \leq i \leq \lambda$, a path connecting a_i to b_i such that the set of λ paths forms a path cover.

In the case where $k = 2$, both problems A and B coincide with the 2HP. Usually a measure of reliability (or fault-tolerance) of an interconnection network is given by the maximum number of nodes which can fail simultaneously without prohibiting the ability of each fault-free node to communicate with all other fault-free nodes. Connectivity of an interconnection graph corresponds to the reliability of the interconnection network which is subject to node failures. It is well-known that connectivity of a graph G was characterized in terms of disjoint paths joining a pair of vertices in G. Thus, one-to-many disjoint paths joining a source s and k distinct sinks $t1, t2, \ldots, tk$ are required. A related work was presented by Park, in 2004 [77].

Another related problem is the *disjoint paths problem* (or DP, for short), which is defined as follows:

Disjoint Paths
Instance: A graph G and pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices of G.
Question: Do there exist paths $P_1, \ldots, P_k$ of G, mutually vertex disjoint, such that P_i joins s_i and t_i $(1 \leq i \leq k)$?

The problem was shown to be NP-complete by Karp [57] if k is a variable part of the input. For fixed k, however, the problem is more tractable. For instance, in [83, 84, 90], there are polynomial algorithms to solve DP with $k = 2$. In contrast, the corresponding question for directed graphs G where we seek directed paths $P_1, \ldots, P_k$ is NP-complete even with $k = 2$ [33]. In 1995, Robertson and Seymour showed that for any fixed k there is a polynomial algorithm to solve DP. Furthermore, as a consequence of their algorithm, there is a polynomial algorithm for the subgraph homeomorphism problem, which is one of Garey and Johnson' s open problems [36].

In [27], Damaschke provided a foundation for obtaining polynomial time algorithms for several problems concerning paths in interval graphs, such as finding Hamiltonian paths and circuits, and partitions into paths. In the same paper, he stated that the complexity status of both 1HP and 2HP problems on interval graphs remains an open question.

Motivated by the above issues, we define other path cover related problems, namely, the k-fixed-endpoint path cover problem, the k-fixed-endpoint-set path cover problem, the 2-terminal-set path cover problem and the $k(2)$-terminal-set path cover problem which are defined and solved for the class of cographs in Chapters 5, 6, 7 and 8, respectively.

The cographs, short for complement reducible graphs, are defined as the class of graphs formed from a single vertex under the closure of the operations of union and complementation, namely: (i) a single-vertex graph is a cograph; (ii) the disjoint union of cographs is a cograph; (iii) the complement of a cograph is a cograph. Cographs were introduced in the early 1970s by Lerchs [59] who studied their structural and algorithmic properties. Along with other properties, Lerchs has shown that they admit a unique tree representation, up to isomorphism, called a cotree. Cographs have arisen in many disparate areas of applied mathematics and computer science and have been independently rediscovered by various researchers under various names such as D^*-graphs [54] and P_4 restricted graphs [22, 24]. They are perfect and in fact form a proper subclass of permutation graphs and distance hereditary graphs; they contain the class of quasi-threshold graphs and, thus, the threshold graphs [13, 40]. Furthermore, they are precisely the graphs which contain no induced subgraph isomorphic to a P_4 (i.e., a chordless path on four vertices). The study of cographs led naturally to constructive characterizations that implied several linear-time recognition algorithms that also enabled the construction of the cotree in linear time [13]. Surprisingly, despite the structural simplicity of cographs, constructing linear-time recognition algorithms has been challenging. The first linear-time recognition and cotree-construction algorithm was proposed by Corneil, Perl, and Stewart [24] in 1985. Recently, Bretscher et al. [14] presented a simple linear-time recognition algorithm which uses a multisweep LexBFS approach; their algorithm either produces the cotree of the input graph or identifies an induced P_4. Additionally, since the cographs are perfect, many interesting optimization problems in graph theory, which are NP-complete in general graphs, have polynomial sequential solutions [13, 40]; for example, for the problem of determining the minimum path cover for a cograph, Lin et al. [61] presented a linear-time algorithm, which can be used to produce a Hamiltonian cycle or path, if such a structure exists. Jung [54] studied the existence of a Hamiltonian path or cycle in a cograph.

Hochstättler and Tinhofer [46] presented a sequential algorithm for the path cover problem on P_4-sparse graphs, which runs in $f(n) + O(n)$ time, where $f(n)$ is the time complexity for the construction of a tree representation of a P_4-sparse graph. The class of P_4-*sparse* graphs is defined as the class which contains the graphs for which every set of five vertices induces at most one chordless path on four vertices [45]. This class has been extensively studied and several sequential and/or parallel algorithms for the recognition and classical optimization problems have been proposed [38]. Giakoumakis et al. in [38] studied hamiltonicity properties for the class of P_4-tidy graphs (a proper superclass of P_4-sparse graphs); see also [13].

For the class of quasi-threshold graphs (a proper subclass of cographs), the problem of recognizing whether such a graph is a Hamiltonian graph and finding a Hamiltonian path (cycle) was solved in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model [72]; in the same work, the coloring and other optimization problems was also solved in $O(\log n)$ time using a linear number of processors. Recently, Hsieh et al. [49] presented an $O(n+m)$-time sequential algorithm for the Hamiltonian problem on a distance-hereditary graph and also proposed a parallel implementation of their algorithm which solves the problem in $O(\log n)$ time using $O((n+m)/\log n)$ processors on a PRAM model. A unified approach to solving the Hamiltonian problems on distance-hereditary graphs was presented in [50], while Hsieh [48] presented an efficient parallel strategy for the 2HP problem on the same class of graphs. Algorithms for the path cover problem on other classes of graphs were proposed in [4, 51, 86].

Recently, Nakano et al. in [70] offered a time- and work-optimal parallel solution for the path cover problem on the class of cographs. In particular, they first proved that any algorithm that solves the path cover problem on a cograph of n vertices represented by its modular decomposition tree must take $\Omega(\log n)$ time on the CREW PRAM model, and then showed that this time lower bound is tight for the class of cographs by presenting an EREW algorithm that, given an n-vertex cograph G represented by its cotree , finds and reports a minimum path cover of G in $O(\log n)$ time using $O(n/\log n)$ processors. It is worth noting that it was open for more than 10 years to find a time- and work-optimal parallel solution for this important problem.

Nakano et al. in [70] use novel techniques that combine in a clever way tree structures, called *path trees*, and sequences of square and round brackets; their algorithm produces a minimum path cover of a cograph by finding matchings of brackets in these sequences, constructing path trees, and converting the path trees to a minimum path cover using the inorder traversal. In [70] they left open the problem of applying their technique into other classes of graphs. Motivated by this issue we generalize their technique and apply it to the class of P_4-sparse graphs. We investigate the structure of the paths that occur in a minimum path cover of a P_4-sparse graph and the structure of the corresponding path trees, and present a time- and work-optimal algorithm that runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. We also show that our results can be extended to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs.

The class of proper interval graphs has also been extensively studied in the literature [40, 80] and several linear-time algorithms are known for their recognition and realization [21, 29, 75]. Both Hamiltonian Circuit (HC) and Hamiltonian Path (HP) problems are polynomially solvable for the class of proper interval graphs. Bertossi [8] proved that a proper interval graph has a Hamiltonian path if and only if it is connected. He also gave an $O(n\log n)$ algorithm for finding a Hamiltonian circuit in a proper interval graph. Panda and Pas [75] presented a linear-time algorithm to detect if a proper interval graph is Hamiltonian.

Interval graphs form an important class of perfect graphs [40] and many problems that are NP-complete on arbitrary graphs are shown to admit polynomial time algorithms on this class [4, 40, 58]. Both Hamiltonian Circuit (HC) and Hamiltonian Path (HP) problems are polynomially solvable for the class of interval graphs. Keil introduced a linear-time algorithm for the HC problem on interval graphs [58] and Arikati and Rangan [4] presented a linear-time algorithm for the minimum path cover problem on interval graphs. Motivated by the above issues and by a work of Damaschke [27], where he stated that the complexity status of both 1HP and 2HP problems on interval graphs remains an open question, we solve the k-fixed-endpoint path cover problem and the 1-fixed-endpoint-path cover problem on proper interval and on interval graphs, respectively.

In this work, we also study the k-path partition problem, a generalization of the path partition problem [36]; the *path partition problem* is to determine the minimum number of paths in a path partition of a simple graph G, while a path partition of G is a collection of vertex disjoint paths $P_1, P_2, \ldots, P_r$ in G whose union is $V(G)$. A path partition is called a *k-path partition* if none of the paths has length more than k, for a given positive integer k. The *k-path partition problem* is to determine the minimum number of paths in a k-path partition of a graph G. It is a natural graph problem with applications in broadcasting in computer and communications networks [87, 92] and it is NP-complete for general graphs [36]. Yan et al. [92] gave a polynomial time algorithm for finding the minimum number of paths in a k-path partition of a tree , while Steiner [88] showed that the problem is NP-complete even for cographs if k is considered to be part of the input, but it is polynomially solvable if k is fixed; he also presented a linear-time solution for the problem, with any k, for threshold graphs. Quite recently, Steiner [87] showed that the k-path partition problem remains NP-complete on the class of chordal bipartite graphs if k is part of the input and on the class of comparability graphs even for $k = 3$. Furthermore, he presented a polynomial time solution for the problem, with any k, on bipartite permutation graphs and left the problem open for the class of convex graphs.

1.4.2 Coloring Problems

A *harmonious coloring* of a simple graph G is a proper vertex coloring such that each pair of colors appears together on at most one edge, while a *pair-complete coloring* of G is a proper vertex coloring such that each pair of colors appears together on at least one edge; the *harmonious chromatic number* $h(G)$ of the graph G is the least integer k for which G admits a harmonious coloring with k colors and its *achromatic number* $\psi(G)$ is the largest integer k for which G admits a pair-complete coloring with k colors.

Harmonious coloring developed from the closely related concept of line-distinguishing coloring which was introduced independently by Frank et al. [34] and by Hopcroft and Krishnamoorthy [47] who showed that the harmonious coloring problem is NP-complete on general graphs. The achromatic number was introduced by Harary et al. [42, 43], while the pair-complete coloring problem was proved to be NP-hard on arbitrary graphs by Yannakakis and Gavril [93]. The complexity of both problems has been extensively studied on various classes of perfect graphs such as cographs, interval graphs, bipartite graphs and trees [13, 40]. Bodlaender [11] provides a proof for the NP-completeness of the pair-complete coloring problem for disconnected cographs and disconnected interval graphs, and extends his results for the connected cases. His proof also establishes the NP-hardness of the harmonious coloring problem for disconnected interval graphs and disconnected cographs. It is worth noting that the problem of determining the harmonious chromatic number of a connected cograph is trivial, since in such a graph each vertex must receive a distinct color as it is at distance at most 2 from all other vertices [15]. Bodlaender's results establish the NP-hardness of the pair-complete coloring problem for the class of permutation graphs and, also, the NP-hardness of the harmonious coloring problem when restricted to disconnected permutation graphs. Extending the above results, Asdre et al. [?] show that the harmonious coloring problem remains NP-complete on connected interval and permutation graphs.

Concerning the class of bipartite graphs and subclasses of this class, Farber et al. [32] show that the harmonious coloring problem and the pair-compete coloring problem are NP-complete for the class of bipartite graphs. In addition, Edwards et al. [30, 31] show that these problems are NP-complete for trees. Their results also establish the NP-completeness of these problems for the classes of convex graphs and disconnected bipartite permutation graphs. However, the complexity of these problems for connected bipartite permutation graphs and biconvex graphs is not straightforward.

Motivated by this issue we prove that the harmonious coloring problem and the pair-complete coloring problem is NP-complete for connected bipartite permutation graphs, and thus, the same holds for the class of biconvex graphs. Moreover, based on Bodlaender's results [11], we show that the pair-complete coloring problem is NP-complete for quasi-threshold graphs and that the harmonious coloring problem is NP-complete for disconnected quasi-threshold graphs. It has been shown that the harmonious coloring problem is polynomially solvable on threshold graphs. In this chapter we show that the pair-complete coloring problem is also polynomially solvable on this class by proposing a simple linear-time algorithm.

Chapter 2

A Time Optimal Parallel Algorithm for the Path Cover Problem on P_4-sparse Graphs

2.1 Introduction

A well studied problem in graph theory with numerous practical applications is to find a minimum number of vertex-disjoint paths of a graph G that cover the vertices of G. This problem, known as the *path cover problem*, finds application in the fields of database design, networks, code optimization among many others (see [2, 36]); it is well-known that the path cover and many of its variants are NP-complete in general graphs [36]. A graph that admits a path cover of size one is referred to as Hamiltonian. Thus, the path cover problem is at least as hard as the problem of deciding whether a graph has a Hamiltonian path.

The study of graphs with few P_4's (chordless paths on four vertices) has practical applications related to examination scheduling and semantic clustering of index terms [24, 59]. These applications have motivated both the theoretical and algorithmic study of the class of cographs, which contain no induced

P_4's. By extending the notion of a P_4-free graph many classes have been obtained by relaxing in various ways the absence of P_4's.

The class of P_4-*sparse* graphs is defined as the class which contains the graphs for which every set of five vertices induces at most one chordless path on four vertices [45]. This class has been extensively studied and several sequential and/or parallel algorithms for the recognition and classical optimization problems have been proposed. Giakoumakis et al. in [38] solved the recognition problem and also the problems of finding the clique number, the stability number and the chromatic number on P_4-sparse graphs in linear sequential time, i.e., in $O(n+m)$ time, where n and m are the number of vertices and edges of the input graph, respectively. Hochstättler and Tinhofer [46] presented a sequential algorithm for the path cover problem on this class of graphs, which runs in $f(n)+O(n)$ time, where $f(n)$ is the time complexity for the construction of a tree representation of a P_4-sparse graph. Sequential algorithms for optimization problems on other related classes of graphs (proper subclasses or superclasses of P_4-sparse graphs) have been also proposed: Lin et al. in [61] proposed an optimal algorithm for the path cover problem on cographs (a proper subclass of P_4-sparse graphs), while Giakoumakis et al. in [38] studied hamiltonicity properties for the class of P_4-tidy graphs (a proper superclass of P_4-sparse graphs); see also [13].

In a parallel environment, the recognition problem on the class of P_4-sparse graphs was studied in [60] and presented a recognition algorithm running in $O(\log^2 n)$ time with $O(\frac{n^2+nm}{\log n})$ processors on the EREW PRAM model. The problem of finding maximum matching on P_4-tidy graphs was examined in [76] and proposed an optimal parallel algorithm for the problem; the algorithm optimally computes a maximum matching of a P_4-tidy graph given its modular decomposition tree. For the class of quasi-threshold graphs (a proper subclass of cographs), the problem of recognizing whether such a graph is a Hamiltonian graph and finding a Hamiltonian path (cycle) was solved in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model [72]; in the same work, the coloring and other optimization problems was also solved in $O(\log n)$ time using a linear number of processors.

Recently, Nakano et al. in [70] offered a time- and work-optimal parallel solution for the path cover problem on the class of cographs . In particular, they first proved that any algorithm that solves the path cover problem on a cograph of n vertices represented by its modular decomposition tree must take $\Omega(\log n)$ time on the CREW PRAM model, and then showed that this time lower bound is tight for the class of cographs by presenting an EREW algorithm that, given an n-vertex cograph G represented by its cotree , finds and reports a minimum path cover of G in $O(\log n)$ time using $O(n/\log n)$ processors. It is worth noting that it was open for more than 10 years to find a time- and work-optimal parallel solution for this important problem.

Nakano et al. in [70] use novel techniques that combine in a clever way tree structures, called *path trees*, and sequences of square and round brackets; their algorithm produces a minimum path cover of a cograph by finding matchings of brackets in these sequences, constructing path trees, and converting the path trees to a minimum path cover using the inorder traversal. In [70] they left open the problem of applying their technique into other classes of graphs. Motivated by this issue we generalize their technique and apply it to the class of P_4-sparse graphs. We investigate the structure of the paths that occur in a minimum path cover of a P_4-sparse graph and the structure of the corresponding path trees, and present a time- and work-optimal algorithm that runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. We also show that our results can be extended to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs.

Our work is organized as follows. In Section 2.2 we establish the notation and related terminology and we present background results. In Section 2.3 we investigate the paths that occur in a minimum path cover of a P_4-sparse graph, while in Section 2.4 we describe the path trees that efficiently produce such paths in a parallel process environment. Section 2.5 describes the construction of the bracket sequences

[70] and in Section 2.6 we describe our optimal parallel path cover algorithm. In Section 2.7 we extend our results to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs. Finally, in Section 2.8 we conclude the chapter and discuss possible future extensions.

2.2 Preliminaries

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively.

2.2.1 Modular Decomposition

A subset M of vertices of a graph G is said to be a *module* of G, if every vertex outside M is either adjacent to all vertices in M or to none of them. The emptyset, the singletons, and the vertex set $V(G)$ are *trivial* modules and whenever G has only trivial modules it is called a *prime* (or *indecomposable*) *graph*. A non-trivial module is also called *homogeneous set* . A module M of the graph G is called a *strong module*, if for any module M' of G, either $M' \cap M = \emptyset$ or one module is included into the other. Furthermore, a module in G is also a module in $\overline{G}$.

The *modular decomposition* of a graph G is a linear-space representation of all the partitions of $V(G)$ where each partition class is a module. The *modular decomposition tree* $T(G)$ of the graph G (or *md-tree* for short) is a unique labelled tree associated with the modular decomposition of G in which the leaves are the vertices of G and the set of leaves associated with the subtree rooted at an internal node induces a strong module of G. Thus, the md-tree $T(G)$ represents all the strong modules of G. An internal node is labelled by either P (for *parallel* module), S (for *series* module), or N (for *neighborhood* module). It is shown that for every graph G the md-tree $T(G)$ is unique up to isomorphism and it can be constructed sequentially in linear time [26, 68]. Dahlhaus in [25] suggests a parallel algorithm for the modular decomposition of graphs that runs in $O(\log^2 n)$ on a CRCW PRAM with a linear number of processors. The approach of Dahlhaus (1995) recursively develops the modular decomposition of two induced subgraphs, which are then spliced together to produce the modular decomposition of the whole graph. Moreover, using $O(n+m)$ EREW processors the modular decomposition tree can be retrieved in $O(\log^3 n)$ as shown in [26].

Let t be an internal node of the md-tree $T(G)$ of a graph G. We denote by $M(t)$ the module corresponding to t which consists of the set of vertices of G associated with the subtree of $T(G)$ rooted at node t; note that $M(t)$ is a strong module for every (internal or leaf) node t of $T(G)$. Let $u_1, u_2, \ldots, u_p$ be the children of the node t of md-tree $T(G)$. We denote by $G(t)$ the *representative graph* of the module $M(t)$ defined as follows: $V(G(t)) = \{u_1, u_2, \ldots, u_p\}$ and $u_i u_j \in E(G(t))$ if there exists an edge $v_k v_\ell \in E(G)$ such that $v_k \in M(u_i)$ and $v_\ell \in M(u_j)$.

By the definition of a module, if a vertex of $M(t_i)$ is adjacent to a vertex of $M(t_j)$ then every vertex of $M(t_i)$ is adjacent to every vertex of $M(t_j)$. Thus $G(t)$ is isomorphic to the graph induced by a subset of $M(t)$ consisting of a single vertex from each maximal strong submodule of $M(t)$ in the modular decomposition of G. It is easy to show that the following lemma holds (see also [39]):

Lemma 2.1. *Let G be a graph, $T(G)$ its modular decomposition tree, and t an internal node of $T(G)$. Then, $G(t)$ is an edgeless graph if t is a P-node, $G(t)$ is a complete graph if t is an S-node, and $G(t)$ is a prime graph if t is an N-node.*

2.2.2 P_4-sparse Graphs

Below, we review characterizations and properties of P_4-sparse graphs and prove results which we use in our algorithm for the solution of the problem of the path cover of a P_4-sparse graph.

A graph G is called a *spider* if the vertex set $V(G)$ of the graph G admits a partition into sets S, K, and R such that:

P1: $|S| = |K| \geq 2$, the set S is a stable set, and the set K is a clique;

P2: all the vertices in R are adjacent to all the vertices in K and to no vertex in S;

P3: there exists a bijection $f : S \longrightarrow K$ such that exactly one of the following statements holds:

(i) for each vertex $v \in S$, $N(v) \cap K = \{f(v)\}$;

(ii) for each vertex $v \in S$, $N(v) \cap K = K - \{f(v)\}$.

The triple (S, K, R) is called the *spider-partition.* A graph G is a *prime spider* if G is a spider with $|R| \leq 1$. If the condition of case P3(i) holds then the spider G is called a *thin spider*, whereas if the condition of case P3(ii) holds then G is a *thick spider*; note that the complement of a thin spider is a thick spider and vice versa. A prime spider with $|S| = |K| = 2$ is simultaneously thin and thick.

It turns out that a P_4-sparse graph contains prime spiders with special properties which will be detailed later. Thus, we need to identify efficiently the spider partition of a prime spider graph G on n vertices and m edges. Based on the structural properties of a prime spider graph G, we can determine its degree sequence and partition its vertices according to the maximum and minimum degree of the sequence. This approach, though, requires $O((n + m)/\log n)$ processors and, thus, it is not optimal. Nevertheless, we prove the following lemma.

Lemma 2.2. *Let G be a prime spider on n vertices. We can recognize whether G is a thin or a thick spider in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. We denote by (S, K, R) the spider-partition of G. Note that if n is an even number we know that $R = \emptyset$; otherwise $R = \{r\}$. Let $n = 2\ell + 1$ (resp. $n = 2\ell$), where $\ell \geq 3$. We choose an arbitrary vertex $x \in V(G)$ and compute the sets of its neighbors $N(x)$ and non-neighbors $\overline{N}(x)$ in G. Based on the fact that G is a prime spider we have the following cases:

(i.1) $|N(x)| = 1$: G is a thin spider, since $x \in S$.

(i.2) $|N(x)| = \ell + 1$ (resp. $|N(x)| = \ell$): G is a thin spider, since $x \in K$.

(i.3) $|N(x)| = \ell - 1$ (resp. $|N(x)| = \ell - 1$): G is a thick spider ($x \in S$).

(i.4) $|N(x)| = 2\ell - 1$ (resp. $|N(x)| = 2\ell - 2$): G is a thick spider ($x \in K$).

(i.5) $|N(x)| = \ell$: In this case, we can not immediately detect whether G is a thin or a thick spider, since $x \in R$, but we have that $K = N(x)$ and $S = \overline{N}(x)$. Thus, we choose a vertex y of the set $\overline{N}(x)$ and check the number $|N(y)|$ by the cases (i.1) and (i.3).

All the cases are verified by the definitions of a thin and a thick spider. Since the sets $N(x)$ and $\overline{N}(x)$ of an n-vertex graph G are computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors, the five cases can be checked within the same time and processor complexity on the same model of computation. ∎

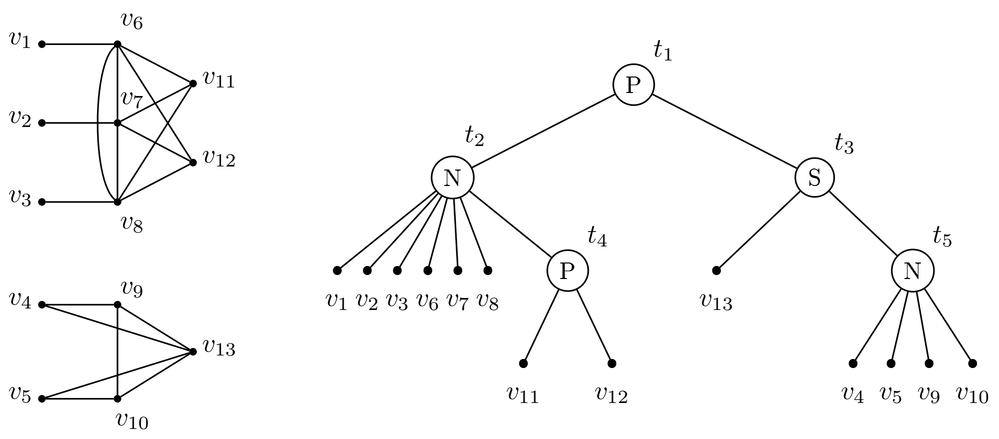


Figure 2.1: A disconnected P_4-sparse graph G on 13 vertices and the corresponding modular decomposition tree $T(G)$.

Let us now return to general P_4-sparse graphs. Let G be a graph and t be an N-node of the md-tree $T(G)$; recall that the vertices of $G(t)$ are the children of node t in $T(G)$. Giakoumakis and Vanherpe [39] showed the following result:

Lemma 2.3. *Let G be a graph and let $T(G)$ be its modular decomposition tree. The graph G is P_4-sparse iff for every N-node t of $T(G)$, $G(t)$ is a prime spider with a spider-partition (S, K, R) and no vertex of $S \cup K$ is an internal node in $T(G)$.*

The above lemma implies that every N-node t of the md-tree $T(G)$ of a P_4-sparse graph G has either $2k$ or $2k+1$ children, where $|S| = |K| = k \geq 2$ and $|R| \leq 1$; the sets S, K, and R form the spider-partition of the graph $G(t)$. More precisely, the N-node t has k children which correspond to S, k children which correspond to K, and either no other child if $R = \emptyset$ or one more child if $R \neq \emptyset$ (in this case, $|R| = 1$ and this child is the root of a subtree of $T(G)$). The children which correspond to S and K are leaves in $T(G)$ and, thus, they are vertices of G, while the child which corresponds to R, if $R \neq \emptyset$, is either a leaf (i.e., a vertex of G) or an internal node labelled by either P, S, or N.

The md-tree $T(G)$ depicted in Fig. 2.1 contains two N-nodes, that is, the nodes t_2 and t_5. The graph $G(t_2)$ is a prime spider on seven vertices with spider-partition $S = \{v_1, v_2, v_3\}$, $K = \{v_6, v_7, v_8\}$, and $R = \{t_4\}$, while $G(t_5)$ is also a prime spider on four vertices with spider-partition $S = \{v_4, v_5\}$, $K = \{v_9, v_{10}\}$, and $R = \emptyset$. The graph $G[M(t_2)]$ is a spider (non prime spider) with $R = \{v_{11}, v_{12}\}$, and the graph $G[M(t_5)]$ is also a spider (prime spider) with $R = \emptyset$.

2.2.3 Number of Paths

Based on the techniques described in [62, 70], we modify the tree $T(G)$: We binarize the tree $T(G)$ in such a way that each of its internal nodes labelled by either P or S has exactly two children; we denote by $T_b(G)$ the resulting tree. The left and right child of an internal P-node or S-node t of $T_b(G)$ will be denoted by t_l and t_r, respectively. Note that if $T(G)$ has only N-nodes then $T_b(G)$ coincides with $T(G)$.

Let $G[M(t)]$ denote the subgraph induced by the leaf descendants of t in $T_b(G)$, and let $L(t)$ denote the number of vertices of $G[M(t)]$. We say that $T_b(G)$ is *leftist*, denoted by $T_{bl}(G)$, if for every internal node t labelled by either P or S, the condition $L(t_l) \geq L(t_r)$ is satisfied, where t_l and t_r are the left and right child of t, respectively. For every S-node t of $T_{bl}(G)$, we replace the subtree rooted at node t_r with the $L(t_r)$ leaves and call the resulting tree the *reduced* leftist binary tree of $T_{bl}(G)$; we denote it by $T_{blr}(G)$.

Let $\lambda(t)$ denote the number of paths in the minimum path cover of the graph $G[M(t)]$. It is easy to see that, in order to construct the path cover using the tree $T_{blr}(G)$, we need to know the number of paths $\lambda(t)$ of each internal node $t \in T_{blr}(G)$. Recall that, if t is a P-node or S-node then it has a left child t_l and a right child t_r; otherwise, t is an N-node and it has at least 4 children which induce a prime spider $G(t) = (S, K, R)$ with either $R = \emptyset$ or $R = \{r\}$, $r \in T_{blr}(G)$. Note that in the case where $R = \emptyset$ we set $\lambda(r) = 0$, otherwise $\lambda(r) \geq 1$. Based on the results of [46, 62], we obtain the following formula for the number of paths in a minimum path cover of a P_4-sparse graph.

$$\lambda(t) = \begin{cases} \lambda(t_l) + \lambda(t_r) & \text{if } t \text{ is a P-node,} \\ \max\{1,\ \lambda(t_l) - L(t_r)\} & \text{if } t \text{ is an S-node,} \\ \lambda(r) + \left\lceil \max\left\{0,\ \frac{|K|-2\lambda(r)}{2}\right\} \right\rceil & \text{if } G(t) \text{ is a thin spider,} \\ \max\{1,\ \lambda(r)\} & \text{if } G(t) \text{ is a thick spider.} \end{cases} \tag{2.1}$$

We conclude with the following results.

Lemma 2.4. *Let G be a P_4-sparse graph on n vertices and let $T(G)$ be its modular decomposition tree. The leftist binary tree $T_{bl}(G)$ and the reduced leftist binary tree $T_{blr}(G)$ can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. Since we only binarize each subtree of $T(G)$ rooted at a P-node or an S-node and since $T(G)$ contains $O(n)$ nodes, we can construct both the leftist binary tree $T_{bl}(G)$ and the reduced leftist binary tree $T_{blr}(G)$ in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors; see [1] and Lemma 5.2 of [70]. ∎

Lemma 2.5. *Let G be a P_4-sparse graph on n vertices and $T_{blr}(G)$ be its reduced leftist binary tree. For every internal node t of $T_{blr}(G)$, the number of paths $\lambda(t)$ in a minimum path cover of $G[M(t)]$ can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

Proof. By Eq. (2.1) and Lemma 2.4 of [70] it suffices to detect whether $G(t)$ is a thin spider or a thick spider for every internal N-node t of $T_{blr}(G)$. Let q be the number of the N-nodes of $T_{blr}(G)$ and let t_i be an internal N-node of $T_{blr}(G)$, $1 \leq i \leq q$. By Lemma 2.2 it follows that each graph $G(t_i)$ can be checked in $O(\log n_i)$ time using $O(n_i/\log n_i)$ EREW PRAM processors, where $n_i = V(G(t_i))$. Recall that for every graph $G(t_i)$ at least $n_i - 1$ vertices are leaves in $T_{blr}(G)$; see Lemma 2.3. Thus the overall time and processor complexity are $O(\log n)$ time and $O(n/\log n)$ EREW PRAM processors, respectively. ∎

2.2.4 Path Trees and Bracket Matching

In this section we review the *path trees* [70], which are the key ingredients of our algorithm and we show that using bracket matching we can make the construction of path trees more efficient.

Let G be a graph and let $P = [p, \ldots, p']$ be a path of a minimum path cover $\mathcal{P}$ of G. Note that, if G is a hamiltonian graph then $\mathcal{P} = \{P\}$. A path tree, denoted by $T(p, p')$, is a rooted binary tree whose nodes are exactly the vertices of a path P of the minimum path cover $\mathcal{P}$ of G and p, p' are the endpoints of P. The vertices of the path tree $T(p, p')$ are placed in $T(p, p')$ in such a way that the inorder traversal of $T(p, p')$ returns the path P. It follows that the path tree of a given path P, is not unique. Note that, a path P can be constructed from its corresponding path tree $T(p, p')$, optimally in parallel, by applying the Euler tour technique [70].

Let $T_{blr}(G)$ be the reduced leftist binary tree obtained from the md-tree $T(G)$ of the graph G. In order to construct the path trees efficiently in a parallel environment, we generate a sequence of square/round brackets for each node of $T_{blr}(G)$. We use two types of brackets: Square brackets "[" and "]" and round brackets "(" and ")". The path trees are constructed by finding matching pairs of square brackets and matching pairs of round brackets independently. Note that, given a bracket sequence corresponding to the vertices of a graph G, the path trees and consequently the path cover of G can be constructed optimally. These matchings correspond to the edges of a path tree. Specifically, for vertices a and b, we establish an edge as follows:

- $\overset{a^p}{[}\ \overset{b^l}{]}$ an edge connecting the vertex a to its parent b as a left child;
- $\overset{a^p}{[}\ \overset{b^r}{]}$ an edge connecting the vertex a to its parent b as a right child;
- $\overset{a^l}{(}\ \overset{b^p}{)}$ an edge connecting the vertex b to its parent a as a left child;
- $\overset{a^r}{(}\ \overset{b^p}{)}$ an edge connecting the vertex b to its parent a as a right child.

2.2.5 Time and Work Optimality

Let G be a cograph on n vertices and let $T(G)$ be its representative cotree. For the minimum path cover problem, Nakano et al. [70] have proved the following result:

Theorem 2.1. *(Nakano et al. [70]): Every algorithm that determines the number of paths in a minimum path cover or reports the minimum path cover of an n-vertex cograph represented by its cotree must take $\Omega(\log n)$ CREW time even if an infinite number of processors is available.*

To verify the time- and work-optimality of our algorithm note that the class of P_4-sparse graphs is a proper superclass of cographs and that the md-tree $T(G)$ of a P_4-sparse graph G contains P-nodes, S-nodes and N-nodes; if $T(G)$ has only P- or S-nodes then $T(G)$ coincides with the definition of a cotree (see [13]). Thus, due to the following result, our algorithm is time- and work-optimal.

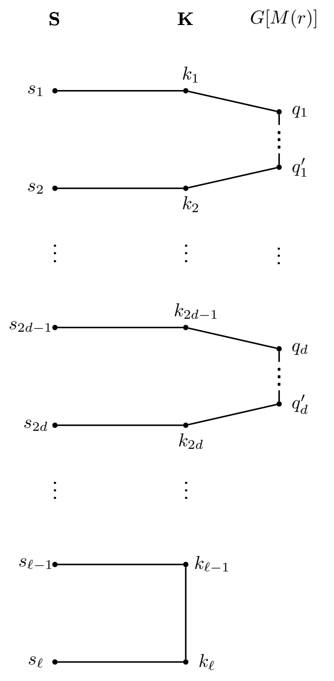
Corollary 2.1. *Every algorithm that determines the number of paths in a minimum path cover or reports the minimum path cover of an n-vertex P_4-sparse graph G represented by its md-tree must take $\Omega(\log n)$ CREW time even if an infinite number of processors is available.*

2.3 Path Cover in P_4-sparse Graphs

In this section we review some ideas for finding a minimum path cover of a P_4-sparse graph G. We suppose that the reduced leftist binarized tree $T_{blr}(G)$ of the input graph G is given. We focus on the internal N-nodes since the cases of the P-nodes and S-nodes have already been studied in [61].

Let t be an internal N-node of $T_{blr}(G)$. Let $\mathcal{P}$ be the minimum path cover of the graph $G[M(t)]$ and let $\lambda(t)$ be the number of paths in $\mathcal{P}$, i.e., $\lambda(t) = |\mathcal{P}|$; recall that, $M(t)$ is the module which corresponds to node t and consists of all the vertices of G associated with the subtree of $T(G)$ rooted at t.

Let $G(t) = (S, K, R)$ be a prime spider and let $S = \{s_1, s_2, \ldots, s_\ell\}$ and $K = \{k_1, k_2, \ldots, k_\ell\}$, where $|S| = |K| = \ell$; by definition, there exists a bijection f such that $f(s_i) = k_i, 1 \leq i \leq \ell$. If $R = \{r\}$ then let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_d\}$ be a minimum path cover of $G[M(r)]$, where $d = \lambda(r)$, and let q_i and q'_i be the endpoints of the path Q_i, $1 \leq i \leq d$. Then, for the computation of the minimum path cover $\mathcal{P}$ of $G[M(t)]$ we distinguish the following two cases.



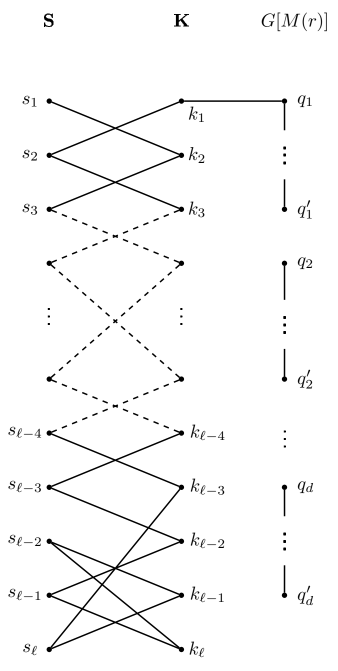


Figure 2.2: Illustrating the path cover of (a) a thin spider graph and (b) a thick spider.

Case 1. $G(t) = (S, K, R)$ is a thin spider.

1.1 $R = \emptyset$. The graph $G[M(t)]$ contains $k = \lceil \frac{\ell}{2} \rceil$ paths. Thus, if ℓ is even then the $\ell/2$ paths in a minimum path cover of $G[M(t)]$ are the following:

$$\mathcal{P} = \{[s_1k_1k_2s_2], [s_3k_3k_4s_4], \dots, [s_{\ell-1}k_{\ell-1}k_\ell s_\ell]\}. \tag{2.2}$$

Note that the form of a path P of $\mathcal{P}$ can be arbitrary in terms of the order of the pair of adjacent vertices s_ik_i; that is, any path P of $\mathcal{P}$ can have the following form $P = [s_ik_ik_js_j]$, for $i \neq j$ and $1 \leq i, j \leq \ell$. For simplicity we adopt the order of the vertices as shown in Eq. (2.2).

In the case where ℓ is odd, the $(\ell + 1)/2$ paths in a minimum path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1k_1k_2s_2], [s_3k_3k_4s_4], \dots, [s_{\ell-2}k_{\ell-2}k_{\ell-1}s_{\ell-1}], [s_\ell k_\ell]\}. \tag{2.3}$$

1.2 $R = \{r\}$. The paths of $G[M(t)]$ are obtained by joining the endpoints of some paths Q_i of $G[M(r)]$ with the vertices of the k paths of $G[S \cup K]$. Thus, if $k \leq d$ (see Fig. 2.2(a)) and ℓ is even we have:

$$\mathcal{P} = \{[s_1k_1q_1 \dots q'_1k_2s_2], \dots, [s_{\ell-1}k_{\ell-1}q_k \dots q'_kk_\ell s_\ell], Q_{k+1}, Q_{k+2}, \dots, Q_d\}, \tag{2.4}$$

while if $k \leq d$ and ℓ is odd we have:

$$\mathcal{P} = \{[s_1k_1q_1 \dots q'_1k_2s_2], \dots, [s_{\ell-2}k_{\ell-2}q_{k-1} \dots q'_{k-1}k_{\ell-1}s_{\ell-1}], [s_\ell k_\ell q_k \dots q'_k], Q_{k+1}, Q_{k+2}, \dots, Q_d\}. \tag{2.5}$$

If $k > d$ and ℓ is even then the paths that occur in a minimum path cover of $G[M(t)]$ are obtained by joining d paths of $G[S \cup K]$ with the d paths of $G[M(r)]$. Thus, we have:

$$\mathcal{P} = \{[s_1k_1q_1 \dots q'_1k_2s_2], \dots, [s_{2d-1}k_{2d-1}q_d \dots q'_dk_{2d}s_{2d}], \dots, [s_{\ell-1}k_{\ell-1}k_\ell s_\ell]\}. \tag{2.6}$$

If $k > d$ and ℓ is odd, then the paths in a minimum path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1 k_1 q_1 \dots q_1' k_2 s_2], \dots, [s_{2d-1} k_{2d-1} q_d \dots q_d' k_{2d} s_{2d}], \dots, [s_{\ell-2} k_{\ell-2} k_{\ell-1} s_{\ell-1}], [s_\ell k_\ell]\}. \tag{2.7}$$

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The graph $G[M(t)]$ is a hamiltonian graph and every edge in the Hamilton path has one endpoint in S and the other endpoint in K (i.e., there exists no Hamilton path which contains an edge with both endpoints in K; for example, see Fig. 2.2(b)). Thus, if ℓ is an odd number and $\ell > 3$ we have:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_{\ell-3} s_{\ell-2} k_{\ell-1} s_\ell k_{\ell-2} s_{\ell-1} k_\ell s_{\ell-3} k_{\ell-4} \dots k_1]\}. \tag{2.8}$$

Note that, if $\ell = 3$ the path is $\mathcal{P} = \{[s_1 k_3 s_2 k_1 s_3 k_2]\}$. In the case where ℓ is even and $\ell = 2$ the graph $G[M(t)]$ is also a thin spider and thus the path that occurs is $\mathcal{P} = \{[s_1 k_2 k_1 s_2]\}$. Thus, if ℓ is even and $\ell > 2$ the paths that occur in a path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_{\ell-2} s_{\ell-1} k_\ell s_{\ell-2} k_{\ell-1} s_\ell k_{\ell-3} s_{\ell-4} k_{\ell-5} \dots k_1]\}. \tag{2.9}$$

We note that in Eqs. (2.8)–(2.9) the order of the vertices can be obtained in many other ways. In fact, if we assume that the set K is an independent set then the Hamilton path of $S \cup K$ can be constructed by any DFS traversal starting from an arbitrary vertex of S. More specifically in Eq. (2.8) (resp. Eq. (2.9)) the order between the vertices s_1 and s_ℓ (resp. k_ℓ) can have the form $k_i s_{i+1} k_j s_{j+1}$, for $i \neq j$, $j \neq i+1$, and $1 < i, j < \ell$. Similarly, between the vertices k_ℓ (resp. s_ℓ) and k_1 we can have an arbitrary order of the vertices of the form $s_i k_{i-1} s_j k_{j-1}$ (resp. $k_i s_{i-1} k_j s_{j-1}$), for $i \neq j$, $j \neq i-1$, and $1 < i, j \leq \ell - 3$. In our study, for convenience we adopt the order shown in Eqs. (2.8)–(2.9).
2.2 $R = \{r\}$. The Hamilton path of $G[S \cup K]$ is connected to the path Q_1 of $G[M(r)]$. Thus, the paths that occur in a path cover of $G[M(t)]$ are:

$$\mathcal{P} = \{[s_1 k_2 s_3 \dots k_3 s_2 k_1 q_1 \dots q_1'], Q_2, Q_3, \dots, Q_d\}. \tag{2.10}$$

Again, if $\ell = 2$ the paths that occur are $\mathcal{P} = \{[s_1 k_2 q_1 \dots q_1' k_1 s_2], Q_2, Q_3, \dots, Q_d\}$, while if $\ell = 3$ we have $\mathcal{P} = \{[s_1 k_3 s_2 k_1 s_3 k_2 q_1 \dots q_1'], Q_2, Q_3, \dots, Q_d\}$.

In both cases, the paths in $\mathcal{P}$ form a minimum path cover of $G[M(t)]$. Note that in a sequential environment we can compute a minimum path cover in a P-node or an S-node t of $T_{blr}(G)$ by using appropriate functions described in [61]. Similar results have appeared in [46].

2.4 Path Trees of P_4-sparse Graphs

Although the sequential algorithm is quite simple, a naive parallelization of this algorithm needs time proportional to the height of the tree $T_{blr}(G)$, which in the worst case is $O(n)$. In order to obtain an efficient parallel algorithm, we make use of the path tree structures and a bracket matching technique introduced in [70] (see Section 2.2).

According to the way that a path tree is constructed, a vertex can be characterized as *insert* or *bridge* vertex . A detailed description of a path tree construction, corresponding to a subtree of $T_{blr}(G)$ rooted at a P-node or an S-node t is presented in [70]. The construction of a path tree corresponding to a subtree of $T_{blr}(G)$ rooted at a P-node or an S-node t is performed by taking the union of two path trees (in the case where t is a P-node) or by inserting a vertex v into a path tree or by using a vertex v to bridge two path trees (in the cases where t is an S-node).

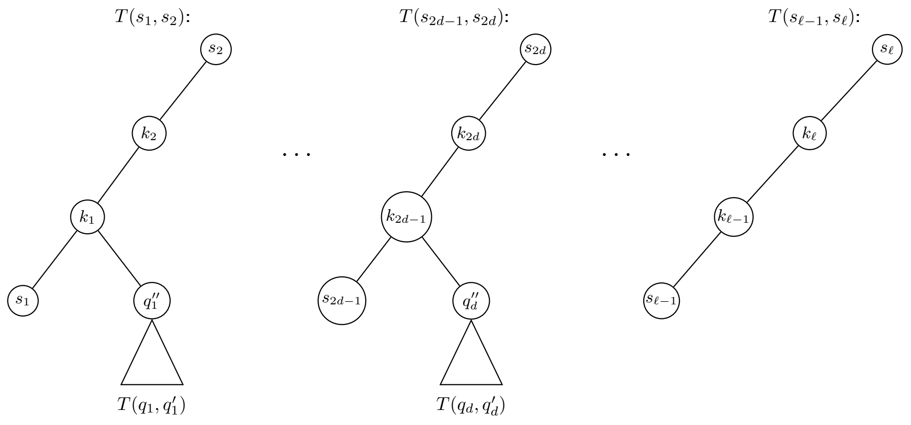


Figure 2.3: The corresponding path trees of Figure 2.2(a).

Let $P_1 = [p_1, \ldots, p_1']$ and $P_2 = [p_2, \ldots, p_2']$ be two paths of a graph G and let $T(p_1, p_1')$ and $T(p_2, p_2')$ be their path trees rooted at nodes $p_1'' \in P_1$ and $p_2'' \in P_2$.

Suppose that a vertex $v \notin P_1$ has to be inserted in the path P_1. In this case, we seek for an appropriate modification of the path tree $T(p_1, p_1')$. Let u be a node of $T(p_1, p_1')$, i.e., $u \in P_1$, having at most one child (either left or right) in $T(p_1, p_1')$. Then, the vertex v is inserted in the path tree $T(p_1, p_1')$ either (i) as a left or right child of the node u of $T(p_1, p_1')$ or (ii) as the root of the path tree $T(p_1, p_1')$ (in this case, p_1'' becomes a left or right child of vertex v).

Suppose now that a vertex $v \notin P_1 \cup P_2$ has to bridge the two paths P_1 and P_2, i.e., v is connected to an endpoint, say, p_1', of P_1 and to an endpoint, say, p_2', of P_2. In this case, the vertex v merges the two path trees $T(p_1, p_1')$ and $T(p_2, p_2')$ into a new path tree having root v with children the roots p_1'' and p_2'' of the path trees $T(p_1, p_1')$ and $T(p_2, p_2')$, respectively.

Next we describe a procedure which generates a path tree corresponding to a subtree of $T_{blr}(G)$ rooted at an N-node. In general, the paths $P_i = [p_i, \ldots, p_i']$ of a path cover $\mathcal{P}$ of a prime spider are considered to be path trees rooted at either p_i or p_i', $1 \leq i \leq |\mathcal{P}|$.

Let G be a P_4-sparse graph and let $T_{blr}(G)$ be its reduced leftist binary tree. Let t be an N-node and let $G(t) = (S, K, R)$ be a prime spider with $S = \{s_1, s_2, \ldots, s_\ell\}$ and $K = \{k_1, k_2, \ldots, k_\ell\}$, where $|S| = |K| = \ell$. In Section 2.8 we have described the form of the paths of a minimum path cover of $G(t)$. Every such path P_i is considered as a path tree rooted at a vertex $x \in S \cup K$, where x is the rightmost vertex of the path P_i, $1 \leq i \leq \lambda(t)$. More specifically, we distinguish two cases:

Case 1. $G(t) = (S, K, R)$ is a thin spider.

1.1 $R = \emptyset$. If ℓ is even then every path P_i, $1 \leq i \leq k$, of the path cover of $G[S \cup K]$ has the following form:

$$P_i = [s_{2i-1} k_{2i-1} k_{2i} s_{2i}], \qquad \text{for } 1 \leq i \leq k,$$

where $k = \lceil \frac{\ell}{2} \rceil$. Then each of the k path trees $T(s_{2i-1}, s_{2i})$ is rooted at vertex s_{2i} and each internal node has only a left child. If ℓ is odd then only the k-th path differs from the previous case; it has the form: $P_k = [s_\ell k_\ell]$. In this case the vertex s_ℓ is the root of the corresponding path tree $T(s_\ell, k_\ell)$ and the root s_ℓ has left child the vertex k_ℓ.

1.2 $R = \{r\}$. Let $q''_1, q''_2, \ldots, q''_d$ be the roots of the path trees $T(q_1, q'_1), T(q_2, q'_2), \ldots, T(q_d, q'_d)$ of the graph $G[M(r)]$, where $d = \lambda(r)$. As described above, every vertex q''_i becomes the right child of vertex k_{2i-1}, $1 \leq i \leq d$, in the path trees (see Fig. 2.3). Note that if ℓ is odd and $d \geq k$, then the root of the corresponding path tree of $G[M(r)]$ becomes the right child of vertex k_ℓ.

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The paths described in Eqs. (2.8)–(2.9) are path trees $T(s_1, k_1)$ rooted at vertex k_1, which is the rightmost vertex of the Hamilton path of $G[S \cup K]$. Each internal node u of $T(s_1, k_1)$ has only a left child which is the previous node of u in the sequences of Eqs. (2.8)–(2.9); that is, vertex s_1 is the leftmost leaf of $T(s_1, k_1)$.

2.2 $R = \{r\}$. In this case, one path of a path cover of $G[M(r)]$ is connected to vertex k_1; see Eq. (2.10). Let $q''_1, q''_2, \ldots, q''_d$ be the roots of the path trees $T(q_1, q'_1), T(q_2, q'_2), \ldots, T(q_d, q'_d)$ of $G[M(r)]$, respectively, where $d = \lambda(r)$. Then the resulting path tree $T(s_1, q'_1)$ is similar to $T(s_1, k_1)$ of the previous case; the only difference is that vertex q''_1 becomes the right child of the root k_1 of $T(s_1, q'_1)$ (see Fig. 2.4).

In all the cases, the structure of the corresponding path trees is verified from the fact that the inorder traversal of the path trees returns the paths described in Eqs. (2.2)–(2.10).

2.5 Bracket Sequence on N-node

Let t be a node of $T_{blr}(G)$ and let $T(q_1, q'_1), T(q_2, q'_2), \ldots, T(q_{\lambda(t)}, q'_{\lambda(t)})$ be the path trees of $G[M(t)]$. We denote by $B(t)$ the bracket sequence of node t (see Section 2.4). Note that a bracket matching of $B(t)$ corresponds to an edge of a path tree $T(q_i, q'_i)$, $1 \leq i \leq \lambda(t)$. Thus, all the bracket matchings of $B(t)$ generate the path trees of $G[M(t)]$.

Let $T(p_i, p'_i)$ be a path tree of $G[M(t)]$ and let a be a vertex of $T(p_i, p'_i)$. Then, we have: If a is the root of $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^p}{[}$ in $B(t)$. If a has only a right child in $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^l}{(}$ in $B(t)$, while if a has no right child in $T(p_i, p'_i)$ then there is an unmatched bracket $\overset{a^r}{(}$ in $B(t)$.

The cases where t is a P-node or an S-node of $T_{blr}(G)$ have been established in [70]. Here we study the case where t is an N-node of $T_{blr}(G)$ and we show the construction of an appropriate bracket sequence $B(t)$ using the above results.

Let t be an N-node of $T_{blr}(G)$ and let $G(t) = (S, K, R)$ be a prime spider with $S = \{s_1, s_2, \ldots, s_\ell\}$ and $K = \{k_1, k_2, \ldots, k_\ell\}$, where $|S| = |K| = \ell$. For simplicity, we associate a dummy node $\hat{t}$ to each N-node t and define $B(\hat{t})$ to be the bracket sequence of the vertices of the set $S \cup K$. In the case where $R = \{r\}$, let $B(r)$ be the bracket sequence of node r. For the bracket sequence $B(t)$ of the N-node t, we have:

$$B(t) = \begin{cases} B(\hat{t}) & \text{if } R = \emptyset, \\ B(r) \cdot B(\hat{t}) & \text{if } R = \{r\}, \end{cases} \tag{2.11}$$

where $B(r) \cdot B(\hat{t})$ denotes the concatenation of $B(r)$ and $B(\hat{t})$. Thus, given the bracket sequence $B(r)$, we need to construct the bracket sequence $B(\hat{t})$.

To simplify our description, for each path tree $T(p_i, p'_i)$ of $G[S \cup K]$, we denote by $\Lambda(i)$ the bracket sequence which has the property that its bracket matching generates the path tree $T(p_i, p'_i)$. We also denote by $\Pi(i)$ the bracket sequence of the unmatched brackets which correspond to the root of $T(p_i, p'_i)$ or to the vertices of $T(p_i, p'_i)$ that do not have a child (left or right) in $T(p_i, p'_i)$. For example, if $T(p_i, p'_i)$ contains only a vertex x, then $\Lambda(i) = \emptyset$ since there is no edge in the path tree, and $\Pi(i) = \overset{x^p}{[} \; \overset{x^l}{(} \; \overset{x^r}{(}$ since x

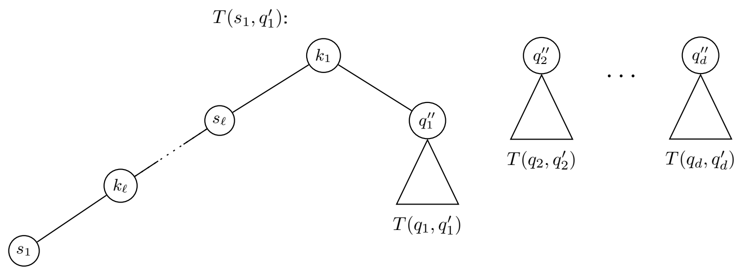


Figure 2.4: The corresponding path trees of Figure 2.2(b).

is the root of the path tree and has neither left nor right child. For the special case where $G[M(t)]$ is a prime spider with $R = \{r\}$, i.e., r is a leaf in $T_{blr}(G)$, we associate the bracket sequence $B(r) = \overset{r^p}{[}\ \overset{r^l}{(}\ \overset{r^r}{(}$. As before, we distinguish two cases for the prime spider $G(t)$.

Case 1. $G(t) = (S, K, R)$ is a thin spider.

1.1 $R = \emptyset$. The graph $G[S \cup K]$ contains $k = \lceil \frac{\ell}{2} \rceil$ paths. Thus, if ℓ is even we associate to node t an appropriate bracket sequence $B(t)$ that generates $k = \frac{\ell}{2}$ path trees $T(q_1, q_1'), T(q_2, q_2'), \ldots, T(q_k, q_k')$. Each path tree $T(q_i, q_i')$ should have the following structure: (i) it consists of four vertices, (ii) it has root a vertex of S, and (iii) its internal vertices have only left children. Recall, that the inorder traversal of $T(q_i, q_i')$ produces the path $P_i = [s_{2i-1}k_{2i-1}k_{2i}s_{2i}]$, which is the i-th path of $\mathcal{P}$ in Eq. (2.2) (see also Section 2.4). The following two bracket sequences generate the path tree $T(q_i, q_i')$:

$$\begin{aligned} \Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}, \quad 1 \le i \le \frac{\ell}{2}, \quad \text{and} \\ \Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad 1 \le i \le \frac{\ell}{2}, \end{aligned}$$

By definition, the matching pair $\overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}$ in $\Lambda(i)$ makes vertex s_{2i-1} to be the left child of vertex k_{2i-1}, the matching pair $\overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}$ makes vertex k_{2i-1} to be the left child of vertex k_{2i}, while the matching pair $\overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}$ makes vertex k_{2i} the left child of vertex s_{2i}. The bracket sequence $\Pi(i)$ consists only of left brackets (round or square), because each vertex of the path tree $T(q_i, q_i')$ should be able to have two children and a parent. In detail, the brackets $\overset{s^l_{2i-1}}{(}$ and $\overset{s^r_{2i-1}}{(}$ mean that the vertex s_{2i-1} can have a left and a right child, respectively, since it is the leftmost leaf of the path tree $T(q_i, q_i')$. The vertices k_{2i-1} and k_{2i} already have only left children in $T(q_i, q_i')$ and thus we add brackets $\overset{k^r_{2i-1}}{(}$ and $\overset{k^r_{2i}}{(}$. As the root s_{2i} of the tree $T(q_i, q_i')$ can have a right child and a parent, we add brackets $\overset{s^p_{2i}}{[}$ and $\overset{s^r_{2i}}{(}$ in $\Pi(i)$.

If ℓ is odd, there are $k = \frac{\ell+1}{2}$ paths in the graph $G[M(t)]$. Thus, $\frac{\ell+1}{2}$ path trees are generated by $B(t)$, which produce the paths in Eq. (2.3). As in the previous case, the $\frac{\ell-1}{2}$ path trees $T(q_i, q_i')$ consist of four vertices and the root of the path tree $T(q_k, q_k')$ is the vertex s_ℓ which has vertex k_ℓ as a left child. Thus, we now need to distinguish the bracket sequence that generates path tree $T(q_k, q_k')$ from the rest $\frac{(\ell-1)}{2}$ path trees $T(q_i, q_i')$. Therefore, the following bracket sequences generate the corresponding path trees:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\,, \quad 1 \le i \le \frac{\ell-1}{2}, \\
\Lambda(\frac{\ell+1}{2}) &= \overset{k^p_{\ell}}{[}\ \overset{s^l_{\ell}}{]}\,, \\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}\,, \quad 1 \le i \le \frac{\ell-1}{2}, \quad \text{and} \\
\Pi(\frac{\ell+1}{2}) &= \overset{k^l_{\ell}}{(}\ \overset{k^r_{\ell}}{(}\ \overset{s^p_{\ell}}{[}\ \overset{s^r_{\ell}}{(}\,,
\end{aligned}$$

1.2 $R = \{r\}$. Since the graph $G[S \cup K]$ contains k paths, the paths of $G[M(t)]$ are obtained by joining the endpoints of the paths $Q_1, Q_2, \ldots, Q_k$ of $G[M(r)]$ with the endpoints of the k paths of $G[S \cup K]$. Thus, we need to connect the path trees $T(q_i, q_i')$ corresponding to the paths Q_i of $G[M(r)]$ with the path trees corresponding to the paths of $G[S \cup K]$ in such a way that the inorder traversal of the resulting path trees provides the correct paths. Thus, if $k \le d$ and ℓ is even we have the following bracket sequences:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\,, \quad 1 \le i \le \frac{\ell}{2}, \quad \text{and} \\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}\,, \quad 1 \le i \le \frac{\ell}{2}.
\end{aligned}$$

Comparing $\Lambda(i)$ and $\Pi(i)$ with the corresponding sequences of the previous case where $R = \emptyset$, one can observe that the round bracket $\overset{k^r_{2i-1}}{(}$ does not appear in $\Pi(i)$ but it is appended to $\Lambda(i)$ as square bracket $\overset{k^r_{2i-1}}{]}$. This is because we need to connect the root of a path tree of $G[M(r)]$ as a right child of vertex k_{2i-1} of the i-th path tree.

Recall that, in the case where ℓ is odd there exists a path tree T_{q_k} consisting of two vertices: The root s_ℓ and its left child k_ℓ. Consequently, if $k \le d$, a path tree of $G[M(r)]$ is connected as a left child of vertex k_ℓ of T_{q_k} and, thus, we have the following bracket sequences:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\,, \quad 1 \le i \le \frac{\ell-1}{2}, \\
\Lambda(\frac{\ell+1}{2}) &= \overset{k^p_{\ell}}{[}\ \overset{s^l_{\ell}}{]}\ \overset{k^l_{\ell}}{]}\,, \\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}\,, \quad 1 \le i \le \frac{\ell-1}{2} \quad \text{and} \\
\Pi(\frac{\ell+1}{2}) &= \overset{k^r_{\ell}}{(}\ \overset{s^p_{\ell}}{[}\ \overset{s^r_{\ell}}{(}\,.
\end{aligned}$$

If $k > d$ then the paths that occur in a minimum path cover of $G[M(t)]$ are obtained by joining d paths of $G[S \cup K]$ with the d paths of $G[M(r)]$. The rest $k - d$ paths of $G[S \cup K]$ remain the same. As a result, if $k > d$ and ℓ is even, the bracket $\overset{k^r_{2i-1}}{]}$ appears only in $\Lambda(i)$ where $1 \le i \le d$. Recall that, the bracket $\overset{k^r_{2i-1}}{]}$ means that vertex k_{2i-1} can have a right child which is the root of a path tree corresponding to a path of $G[M(r)]$; for example, see Fig. 2.5. Therefore, we have the following bracket sequences:

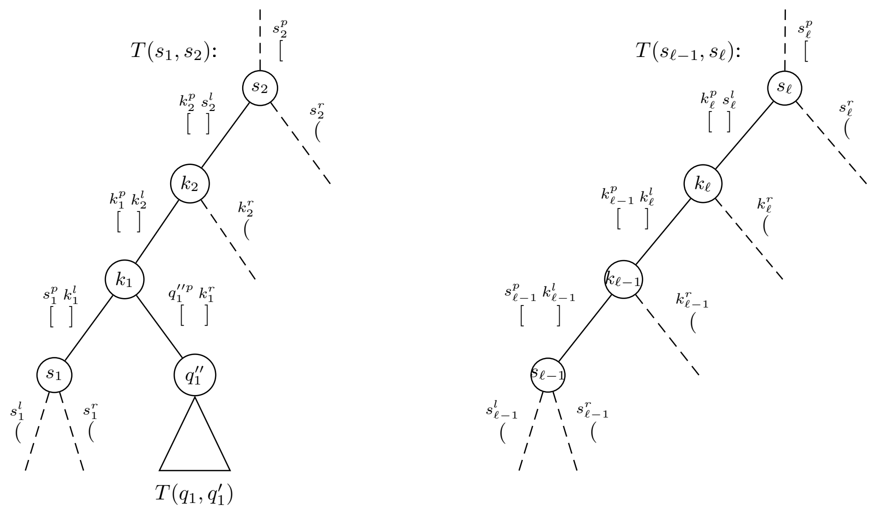


Figure 2.5: The brackets for the path trees $T(s_1, s_2)$ and $T(s_{\ell-1}, s_\ell)$ of Figure 2.3.

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\ , \quad 1 \le i \le d,\\
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}, \quad d < i \le \frac{\ell}{2},\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad 1 \le i \le d \text{ and}\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad d < i \le \frac{\ell}{2}.
\end{aligned}$$

In the case where $k > d$ and ℓ is odd, the sequence of brackets can be generated in a similar way; that is, d paths trees corresponding to paths of $G[S \cup K]$ are joined with the d path trees corresponding to paths of $G[M(r)]$. The rest $k - d$ path trees corresponding to paths of $G[S \cup K]$ remain the same. Recall that, when ℓ is odd there is a path tree generating a path of $G[S \cup K]$ which consists of only two vertices. In order to obtain the paths of $G[M(t)]$ we have the following bracket sequences:

$$\begin{aligned}
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}\ \overset{k^r_{2i-1}}{]}\ , \quad 1 \le i \le d,\\
\Lambda(i) &= \overset{s^p_{2i-1}}{[}\ \overset{k^l_{2i-1}}{]}\ \overset{k^p_{2i-1}}{[}\ \overset{k^l_{2i}}{]}\ \overset{k^p_{2i}}{[}\ \overset{s^l_{2i}}{]}, \quad d < i \le \frac{\ell-1}{2},\\
\Lambda(\frac{\ell+1}{2}) &= \overset{k^p_\ell}{[}\ \overset{s^l_\ell}{]},\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad 1 \le i \le d,\\
\Pi(i) &= \overset{s^l_{2i-1}}{(}\ \overset{s^r_{2i-1}}{(}\ \overset{k^r_{2i-1}}{(}\ \overset{k^r_{2i}}{(}\ \overset{s^p_{2i}}{[}\ \overset{s^r_{2i}}{(}, \quad d < i \le \frac{\ell-1}{2}, \quad \text{and}\\
\Pi(\frac{\ell+1}{2}) &= \overset{k^l_\ell}{(}\ \overset{k^r_\ell}{(}\ \overset{s^p_\ell}{[}\ \overset{s^r_\ell}{(}\ .
\end{aligned}$$

Collecting all the previous results in both Case 1.1 and Case 1.2, we have that for ℓ even the bracket sequence that generates the path trees of $G[S \cup K]$ is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \Lambda(2) \cdots \Lambda(\frac{\ell}{2} - 1) \cdot \Lambda(\frac{\ell}{2}) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\frac{\ell}{2} - 1) \cdot \Pi(\frac{\ell}{2}), \tag{2.12}$$

and for ℓ odd the bracket sequence is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \Lambda(2) \cdots \Lambda(\frac{\ell-1}{2}) \cdot \Lambda(\frac{\ell+1}{2}) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\frac{\ell-1}{2}) \cdot \Pi(\frac{\ell+1}{2}). \tag{2.13}$$

We note that the order between any $\Lambda(i)$ and $\Lambda(j)$ in Eqs. (2.12)–(2.13) does not affect the correctness of the construction of the corresponding path trees. The same holds for any bracket sequences $\Pi(i)$ and $\Pi(j)$. Furthermore the number of brackets of each $\Lambda(i)$ and $\Pi(i)$ is at most seven and six, respectively.

Case 2. $G(t) = (S, K, R)$ is a thick spider.

2.1 $R = \emptyset$. The graph $G[S \cup K]$ is Hamiltonian and a Hamilton path can be constructed by edges which have one endpoint in S and the other endpoint in K. Therefore, the sequence of brackets described below result to a path tree. The first edge of the associated path is the one connecting the first vertex of S, say, s_1, with the second vertex of K, say, k_2. The second edge is that connecting the vertex k_2 with the vertex s_3, while the third edge connects the vertex s_3 with the vertex k_4, and so on. Recall that, by definition there is no edge from a vertex s_j of S to a vertex k_j of K. Also, note that the graph $G[S \cup K]$ does not have a unique Hamilton path; in fact every edge of a Hamilton path can have its endpoints in two arbitrary vertices s_i and k_j as long as $i \neq j$ and $1 < i, j \leq \ell - 3$.

As described in Section 2.3, we need to distinguish two cases depending on the value of ℓ. If ℓ is an odd number, then we have the following bracket sequences:

$$\begin{array}{rcl}
\Lambda(1) & = & \overset{s_1^p}{[}, \quad \Lambda(\ell) = \overset{k_1^l}{]}, \\
\Lambda(i) & = & \overset{k_i^l}{]} \; \overset{k_i^p}{[} \; \overset{s_{i+1}^l}{]} \; \overset{s_{i+1}^p}{[}, \quad i = 2, 4, \ldots \ell - 5, \ell - 3, \\
\Lambda(i) & = & \overset{k_i^l}{]} \; \overset{k_i^p}{[} \; \overset{s_{i-1}^l}{]} \; \overset{s_{i-1}^p}{[}, \quad i = 3, 5, \ldots \ell - 6, \ell - 4, \\
\Lambda_{\text{mid}} & = & \overset{k_{\ell-1}^l}{]} \; \overset{k_{\ell-1}^p}{[} \; \overset{s_\ell^l}{]} \; \overset{s_\ell^p}{[} \; \overset{k_{\ell-2}^l}{]} \; \overset{k_{\ell-2}^p}{[} \; \overset{s_{\ell-1}^l}{]} \; \overset{s_{\ell-1}^p}{[} \; \overset{k_\ell^l}{]} \; \overset{k_\ell^p}{[} \; \overset{s_{\ell-3}^l}{]} \; \overset{s_{\ell-3}^p}{[}, \\
\Lambda_{\text{even}} & = & \Lambda(2) \cdot \Lambda(4) \cdots \Lambda(\ell - 5) \cdot \Lambda(\ell - 3), \\
\Lambda_{\text{odd}} & = & \Lambda(\ell - 4) \cdot \Lambda(\ell - 6) \cdots \Lambda(5) \cdot \Lambda(3),
\end{array}
\qquad
\begin{array}{rcl}
\Pi(1) & = & \overset{s_1^l}{(} \; \overset{s_1^r}{(}, \\
\Pi(i) & = & \overset{s_i^r}{(} \; \overset{k_i^r}{(}, \quad 2 \leq i \leq \ell - 1, \\
\Pi(\ell) & = & \overset{k_1^p}{[} \; \overset{k_1^r}{(}.
\end{array}$$

The sequences $\Lambda(1)$ and $\Lambda(\ell)$ are used to identify the endpoints s_1 and k_1 of the Hamilton path. Each $\Lambda(i)$ consists of one matching pair of vertices k_i and $s_{i'}$, $i' \neq i$, such that the vertex k_i will become the left child of the vertex $s_{i'}$ in the corresponding path tree as described in Section 2.4. By concatenating two sequences $\Lambda(i) \cdot \Lambda(j)$ in the sequence Λ_{even}, where i and j are even numbers, the vertex $s_{i'}$ becomes the left child of the vertex k_j in the corresponding path tree. The same holds for the sequence Λ_{odd}, where, in this case, i and j are odd numbers. In this way we eventually connect the two edges $k_i s_{i'}$ and $k_j s_{j'}$ by adding the edge $s_{i'} k_j$.

Thus, the sequence $\Lambda(1) \cdot \Lambda_{\text{even}}$ constructs a path consisting of the odd-labelled vertices s_j and the even-labelled vertices k_i, $1 \leq i, j \leq \ell - 2$ (where i is even and j is odd). Let s_j be the endpoint of the path produced by the sequences $\Lambda(1) \cdot \Lambda_{\text{even}}$ having an odd label, $1 < j \leq \ell - 2$. Then, we know that we can append a closing square bracket] corresponding to the even-labelled vertex $k_{\ell-1}$ since there is no bracket corresponding to the vertex $s_{\ell-1}$ in the sequence Λ_{even}. Now there is a matching between the bracket of the vertex $k_{\ell-1}$ and the bracket of the vertex s_j. The five matching pairs of brackets in

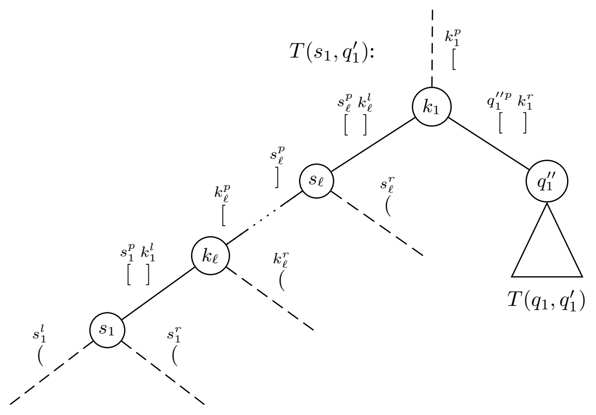


Figure 2.6: The brackets for the path tree $T(s_1, q_1')$ of Figure 2.4.

Λ_{mid} construct the sequence of the corresponding vertices described in Eq. (2.8). In a similar manner the sequence $\Lambda_{\text{odd}} \cdot \Lambda(\ell)$ constructs a path of the even-labelled vertices s_i and the odd-labelled vertices k_j, $1 \leq i, j < \ell - 3$ (where i is even and j is odd).

Note that the above sequences of brackets $\Lambda(i)$ will eventually produce a path tree which has nodes that have only a left child. In order to make each internal node able to connect with a right child we append $\Pi(i)$ which contains right brackets (round or square). In addition, vertex s_1 which is the leftmost leaf of the path tree can obtain both a left and a right child and, therefore, we use the brackets $\overset{s_1^l}{(}\ \overset{s_1^r}{(}$ in $\Pi(1)$. In addition, we use the square bracket $\overset{k_1^p}{[}$ in $\Pi(\ell)$ because vertex k_1 is the root of the path tree and it can become a child itself. Thus, the bracket sequence that generates the path trees of $G[M(t)]$ is the following:

$$B(\hat{t}) = \Lambda(1) \cdot \ \Lambda_{\text{even}} \cdot \Lambda_{\text{mid}} \cdot \Lambda_{\text{odd}} \ \cdot \Lambda(\ell) \cdot \Pi(1) \cdot \Pi(2) \cdots \Pi(\ell). \tag{2.14}$$

In the case where ℓ is even, we have the following bracket sequences:

$$\begin{aligned}
\Lambda(1) &= \overset{s_1^p}{[}, \quad \Lambda(\ell) = \overset{k_1^l}{]}, & \Pi(1) &= \overset{s_1^l}{(}\,\overset{s_1^r}{(}, \\
\Lambda(i) &= \overset{k_i^l}{]}\,\overset{k_i^p}{[}\,\overset{s_{i+1}^l}{]}\,\overset{s_{i+1}^p}{[}, \quad i = 2, 4, \ldots \ell-4, \ell-2, & \Pi(i) &= \overset{s_i^r}{(}\,\overset{k_i^r}{(}, \quad 2 \leq i \leq \ell-1, \\
\Lambda(i) &= \overset{k_i^l}{]}\,\overset{k_i^p}{[}\,\overset{s_{i-1}^l}{]}\,\overset{s_{i-1}^p}{[}, \quad i = 3, 5, \ldots \ell-5, \ell-3, & \Pi(\ell) &= \overset{k_1^p}{[}\,\overset{k_1^r}{(}. \\
\Lambda_{\text{mid}} &= \overset{k_\ell^l}{]}\,\overset{k_\ell^p}{[}\,\overset{s_{\ell-2}^l}{]}\,\overset{s_{\ell-2}^p}{[}\,\overset{k_{\ell-1}^l}{]}\,\overset{k_{\ell-1}^p}{[}\,\overset{s_\ell^l}{]}\,\overset{s_\ell^p}{[}, \\
\Lambda_{\text{even}} &= \Lambda(2) \cdot \Lambda(4) \cdots \Lambda(\ell-4) \cdot \Lambda(\ell-2), \\
\Lambda_{\text{odd}} &= \Lambda(\ell-3) \cdot \Lambda(\ell-5) \cdots \Lambda(5) \cdot \Lambda(3),
\end{aligned}$$

The above sequences are similar to those of the case where ℓ is an odd number. The only difference is in sequence Λ_{mid} in which the three matching pairs can easily be verified by Eq. (2.9). As described above, we need to make each internal node able to connect with a right child, and therefore we append to the sequence $\Lambda(1) \cdot \Lambda_{\text{even}} \cdot \Lambda_{\text{mid}} \cdot \Lambda_{\text{odd}}$ the bracket sequences $\Pi(i)$, $1 \leq i \leq \ell$. Thus, the bracket sequence $B(\hat{t})$ of Eq. (2.14) generates the path trees of $G[S \cup K]$.

2.2 $R = \{r\}$. The Hamilton path of $G[S \cup K]$ is connected to the path Q_1 of $G[M(r)]$. Consequently, the bracket sequences $\Lambda(i)$ and $\Pi(i)$ are similar to those described above in the case where R is empty (for example, see Fig. 2.6). The only difference is that we need bracket $\overset{k_1^r}{]}$ in order to make the path tree corresponding to the path Q_1 of $G[M(r)]$ be the right child of the root vertex, that is vertex k_1, of the path tree corresponding to the Hamilton path of $G[S \cup K]$. Hence, only the two following brackets sequences are different from the previous cases:

$$\Lambda(\ell) \;=\; \overset{k_1^l}{]}\;\overset{k_1^r}{]}\,, \qquad\qquad \text{and} \qquad\qquad \Pi(\ell) \;=\; \overset{k_1^p}{[}\;.$$

Note that in the sequence of brackets $\Pi(\ell)$ we now have one less bracket, compared with the case where R is empty. The missing bracket is $\overset{k_1^r}{(}$ because vertex k_1 already has a right child. As a result, we have that the bracket sequence $B(\hat{t})$ that generates the path trees of $G[S \cup K]$ is described in Eq. (2.14).

We note that the sequences $\Lambda(2)$, $\Lambda(4)$, $\ldots$, $\Lambda(\ell-4)$, $\Lambda(\ell-2)$ can appear in any order in the bracket sequence Λ_{even}; similarly, the sequences $\Lambda(\ell-3)$, $\Lambda(\ell-5)$, $\ldots$, $\Lambda(5)$, $\Lambda(3)$ can appear in any order in Λ_{odd}. The same holds for the sequences $\Pi(i)$, $1 \le i \le \ell$; that is, they can appear in any order in $B(\hat{t})$ after the sequence $\Lambda(\ell)$. Furthermore, the sequences $\Lambda(1)$, Λ_{mid}, $\Lambda(\ell)$, $\Pi(1)$ and $\Pi(\ell)$ contain a constant number of brackets.

2.6 The Algorithm

In this section we present an optimal parallel algorithm for the minimum path cover problem on P_4-sparse graphs. Our algorithm takes as input a P_4-sparse graph G on n vertices and its modular decomposition tree $T(G)$, and finds the paths of a minimum path cover in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

Let us first sketch the workings of our algorithm. Initially, we compute the binarized md-tree $T_b(G)$; recall that we only binarize each subtree of $T(G)$ rooted at a P-node or an S-node. In order to make the binarized tree leftist, we compute the number $L(t)$ for each internal node t of $T_b(G)$. We next compute the number $\lambda(t)$ of paths in the minimum path cover of $G(M[t])$ for each internal node t of $T_{bl}(G)$. Before assigning any bracket sequence, we first compute the reduced leftist binarized tree $T_{blr}(G)$, and, then, for every internal N-node t of $T_{blr}(G)$ we compute a bracket sequence $B(t)$ based on the vertices of $S \cup K$ which are leaves and children of t in $T_{blr}(G)$. Using this information, we generate a bracket sequence $B(t_{root})$ of the root t_{root} of $T_{blr}(G)$. Finally, we construct the path trees by finding all matchings of $B(t_{root})$ and then we return vertex sequences produced by the inorder traversal of the path trees. Recall that the latter corresponds to a minimum path cover of G. We next give the detailed description of the algorithm.

We mention here that Step 4 needs a post-processing function. The bracket assignment procedure for an S-node, as described in [70], may lead to path trees which result to paths having edges that do not appear in the graph G. This post-processing function corrects any *illegal* path tree which has been produced by the children of the S-node. The correct path trees are calculated with a detailed technique, as a post-processing step, proposed in [70].

Time and Processor Complexity. Next, we analyze the time and processor complexity of the proposed algorithm on the PRAM model; for details on PRAM techniques, see [52, 79]. We assume that the input graph G and its modular decomposition tree $T(G)$ are given in adjacency list representations.

Parallel_Minimum_Path_Cover

Input: A P_4-sparse graph G and its modular decomposition tree $T(G)$;

Output: A minimum path cover of the P_4-sparse graph G;

```
1. Compute the binarized tree $T_b(G)$ of $T(G)$ and the number $L(t)$ for each internal node $t$ of $T_b(G)$,
   and then the leftist binarized tree $T_{bl}(G)$;
2. Compute the number of paths $\lambda(t)$ in the minimum path cover of $G[M(t)]$ for each internal node $t$
   of $T_{bl}(G)$, and then the reduced leftist binarized tree $T_{blr}(G)$;
3. Generate the sequence of brackets $B(t_{root})$ of the root $t_{root}$ of $T_{blr}(G)$ based on [70] (for P-nodes
   or S-nodes) and on Eqs (2.11)–(2.14) of Section 2.5 (for N-nodes);
4. Construct the path trees by finding all matchings of $B(t_{root})$;
5. Return a minimum path cover from the path trees;
```

Algorithm 1: Parallel_Minimum_Path_Cover

Steps 1 and 2 are executed in $O(\log n)$ time using $O(n/\log n)$ EREW processors (see Lemma 2.4 and Lemma 2.5). Step 3 computes the sequence of brackets $B(t_{root})$ of the root t_{root} of $T_{blr}(G)$. This computation can be efficiently done by first computing the bracket sequence $B(t)$ of each internal node t of $T_{blr}(G)$ (see results of [70] and Eqs (2.11)–(2.14)), and then contracting the tree $T_{blr}(G)$ into a three-node tree consisting of the root and two nodes. We use standard parallel techniques (i.e., prefix sums, array packing, concatenation) to construct the bracket sequences in each internal node, and the *rake* operation to reduce the tree $T_{blr}(G)$ into a three-node tree (see [52, 79]). Thus, this step can be computed in $O(\log n)$ time using $O(n/\log n)$ EREW processors. Step 4 can be performed in $O(\log n)$ time using $O(n/\log n)$ EREW processors by finding the matching pairs of the sequence $B(t_{root})$, where t_{root} is the root of $T_{blr}(G)$. Note that $B(t_{root})$ has $O(n)$ brackets since the tree $T_{blr}(G)$ has $O(n)$ nodes, and also that each assignment of bracket of an internal node $u \in T_{blr}(G)$ does not exceed the number of its children in $T_{blr}(G)$. Step 5 is accomplished with Euler-tours on the path trees. Since the nodes of the path trees are the vertices of the input graph G, it follows that they contain n nodes, and, thus, Step 5 can be executed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

By Corollary 2.1, the path cover problem of an n-vertex P_4-sparse graph represented by its md-tree must take $\Omega(\log n)$ time on the CREW PRAM model. Our algorithm Parallel_Minimum_Path_ Cover shows that this time lower bound is tight for the class of P_4-sparse graphs. Summarizing, we obtain the following result.

Theorem 2.2. *Let G be a P_4-sparse graph on n vertices and let $T(G)$ be its modular decomposition tree. A minimum path cover of G can be computed time- and work-optimally in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.*

2.7 Other Classes of Graphs

In this section we extend our results to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs. The class of P_4-tidy graphs was introduced by I. Rusu in order to illustrate the notion of P_4-domination in perfect graphs; see [38].

A graph G is *P_4-tidy* if for any induced P_4, say, $abcd$, there exists at most one vertex $v \in V(G) - \{a, b, c, d\}$ such that the subgraph $G[\{a, b, c, d, v\}]$ has at least two P_4's (i.e., the P_4 has at most one *partner*). The P_4-tidy graphs strictly contain the cographs, P_4-reducible, P_4-sparse, P_4-extendible, and P_4-lite graphs. The P_4-lite graphs were defined by Jamison and Olariu in [53]: A graph G is *P_4-lite* if every induced subgraph H of G with at most six vertices either contains at most two P_4's, or is a 3-sun, or is the complement of a 3-sun (a *3-sun* is a thick spider on six vertices with $R = \emptyset$). They remark that every P_4-sparse graph is P_4-lite and prove that every P_4-lite graph is brittle and, thus perfect. We mention here that the P_4-lite graphs coincide with the C_5-free P_4-tidy graphs.

The modular decomposition of P_4-tidy graphs has a structural property, as in the case of P_4-sparse graphs (Lemma 2.3), which is shown by the following result (Theorem 3.2 in [38]):

Theorem 2.3. *(Giakoumakis et al. [38]): Let G be a graph and let $T(G)$ be its modular decomposition tree. The graph G is P_4-tidy iff for every N-node t of $T(G)$, $G(t)$ is either*

(i) a P_5, a $\overline{P_5}$, or a C_5, and no vertex of $G(t)$ is an internal node in $T(G)$, or

(ii) a prime spider (S, K, R) with at most one vertex of $S \cup K$ which is an internal node having two children which are leaves in $T(G)$.

The above theorem implies that every N-node t of the md-tree $T(G)$ of a P_4-tidy graph G has either five vertices which are leaves in $T(G)$ and $G(t) \in \{P_5, \overline{P_5}, C_5\}$ or $G(t) = (S, K, R)$ is a prime spider with at most one vertex, say, t, of $S \cup K$ replaced by a $2K_1$ or a K_2 (i.e., t is a P- or S-node with two children which are leaves in $T(G)$). Based on this result, the path cover problem for the class of P_4-tidy graphs was solved in sequential linear time by describing the paths that occur in every internal node of $T(G)$ [38].

Let G be a P_4-tidy graph, $T(G)$ be its md-tree and t be an N-node of $T(G)$. Here we describe the bracket sequence $B(t)$ that generates the corresponding path trees which produce the paths of a minimum path cover of $G[M(t)]$. According to Theorem 2.3 we distinguish the following cases.

Case A1. $G(t)$ is a P_5, a $\overline{P_5}$ or a C_5.

It is easy to see that $\lambda(t) = 1$, since a path on five vertices (not necessarily chordless) occurs in the three possible graphs (see also [38]). Thus, $\lambda(t)$ can be computed optimally and, hence, it is possible to construct optimally the tree $T_{blr}(G)$. Let $v_1v_2v_3v_4v_5$ be such a path of $G[M(t)]$. Then the corresponding path tree $T(v_1, v_5)$ is constructed as in the case where we have a thick spider with $R = \emptyset$ (Section 2.4, Case 2.1). Thus, it can be viewed as the Hamilton path of a thick spider and the corresponding bracket sequence $B(t)$ is given in details in Section 2.5, Case 2.1.

Case A2. $G(t) = (S, K, R)$ is a prime spider.

In this case and if no vertex of $S \cup K$ is replaced by an S_2 or a K_2, the paths of a minimum path cover of $G[M(t)]$ are obtained by considering the cases of a prime spider of a P_4-sparse graph (Section 2.3). Here, we also have to consider the fact that a vertex of $S \cup K$ can be substituted by an S_2 or a K_2 which will eventually change the value of $\lambda(t)$. Notice that in any case we have to distinguish whether we have a thin or a thick spider and whether the set $R = \emptyset$ or not. For example, if $G(t)$ is a thick spider with $R = \emptyset$ and a vertex of S is replaced by the two vertices of S_2, say, s_1 and s_1', then the paths of a minimum path cover of $G[M(t)]$ are a Hamilton path (see Section 2.3) and an isolated vertex (one of the two vertices, say, s_1', of the set S_2). Thus, in this case, the value of $\lambda(t)$ is equal to the value of the case where $G(t)$ is a thick spider (see Eq. 2.1) plus one.

In a similar manner, we can consider all the other cases and establish the corresponding paths of a minimum path cover of $G[M(t)]$; details about the $\lambda(t)$ and the paths in a minimum path cover of $G[M(t)]$ can be found in [38]. In order to obtain the paths of $G[M(t)]$ we can construct the path trees and the bracket sequence $B(t)$ in a way similar to that described in Section 2.5.

Based on the above description, we can show the following result for the class of P_4-tidy graphs: Given a P_4-tidy graph G on n vertices and its modular decomposition tree $T(G)$, a minimum path cover of G can be optimally computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

2.8 Concluding Remarks

We have presented an optimal parallel algorithm for solving the minimum path cover problem on P_4-sparse graphs; our algorithm runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model, and thus is time- and work-optimal due to the results of [70]. We also described the way we can solve the same problem on the class of P_4-tidy graphs, which forms a proper superclass of P_4-sparse graphs.

An interesting open question would be to see if similar techniques can be efficiently used for finding a minimum path cover for other classes of graphs and solving other related algorithmic problems such as the *terminal path cover* problem: Given a graph G and a subset S of its vertices, the terminal path cover problem is to find a minimum path cover $\mathcal{P}$ of the graph G such that all the vertices of S are endpoints of the paths in $\mathcal{P}$.

Chapter 3

NP-completeness Results for the k-Path Partition Problem

3.1 Introduction

We study the k-path partition problem, a generalization of the path cover problem [36]; recall that the path cover problem is to determine the minimum number of paths in a path cover of a simple graph G, while a path cover of G is a collection of vertex disjoint paths $P_1, P_2, \dots, P_r$ in G whose union is $V(G)$. A path cover is called a *k-path partition* if none of the paths has length more than k, for a given positive integer k. The *k-path partition problem* is to determine the minimum number of paths in a k-path partition of a graph G. It is a natural graph problem with applications in broadcasting in computer and communications networks [87, 92] and it is NP-complete for general graphs [36]. Yan et al. [92] gave a polynomial time algorithm for finding the minimum number of paths in a k-path partition of a tree , while Steiner [88] showed that the problem is NP-complete even for cographs if k is considered to be part of the input, but it is polynomially solvable if k is fixed; he also presented a linear-time solution for the problem, with any k, for threshold graphs. Quite recently, Steiner [87] showed that the k-path partition problem remains NP-complete on the class of chordal bipartite graphs if k is part of the input and on the class of comparability graphs even for $k = 3$. Furthermore, he presented a polynomial time solution for the problem, with any k, on bipartite permutation graphs and left the problem open for the class of convex graphs.

Motivated by Steiner's work [87], we prove that the k-path partition problem is NP-complete on convex graphs. Furthermore, we show that this problem is NP-complete for quasi-threshold graphs, and

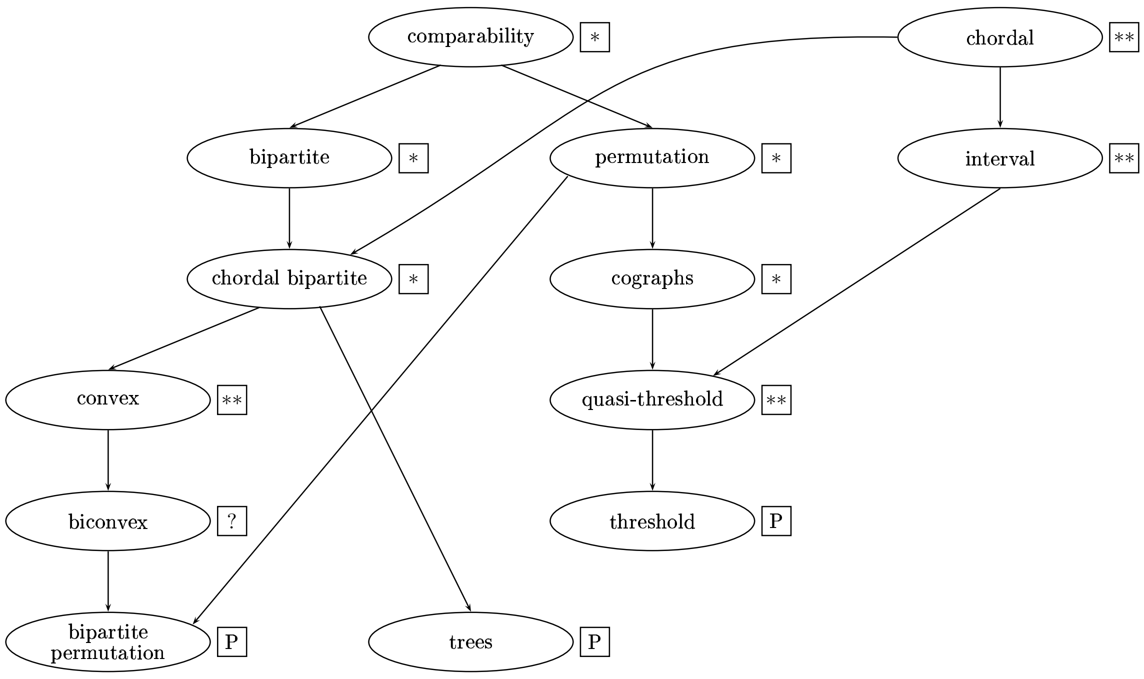


Figure 3.1: The complexity status of the k-path partition problem for some graph subclasses of comparability and chordal graphs. $A \to B$ indicates that class A contains class B. ($*$): NP-complete, previously known; ($**$): NP-complete, new result; (P): polynomial, previously known; (?): unknown.

thus, it is also NP-complete for interval and chordal graphs. For some graph classes, the complexity status of the k-path partition problem is illustrated in Fig. 3.1[1].

This chapter is organized as follows. In Section 3.2 we show that the k-path partition problem is NP-complete on convex graphs, a superclass of bipartite permutation graphs. In Section 3.3 we present structural properties of the class of quasi-threshold graphs and NP-completeness results on this class, while Section 3.4 concludes the chapter and discusses open problems.

3.2 Convex Graphs

We next prove that the k-path partition problem is NP-complete for *convex graphs*; recall that a bipartite graph $G = (X, Y; E)$ is convex on the vertex set X if X can be ordered so that for each element y in the vertex set Y the elements of X connected to y form an interval of X [63].

Theorem 3.1. *The k-path partition problem is NP-complete for convex graphs.*

Proof. The k-path partition problem is obviously in NP. In order to prove NP-hardness , we use a transformation from Bin-Packing. The formulation of the Bin-Packing problem ([SR1] in [36]) is presented below.

[1]Figure 3.1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status for the k-path partition problem on these classes; for definitions of the classes shown, see [13, 40].

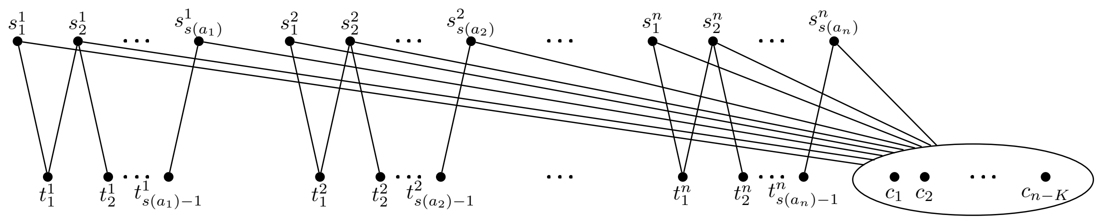


Figure 3.2: Illustrating the constructed convex graph G.

Bin-Packing
Instance: Finite set U of items, a size $s(u) \in Z^+$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K.
Question: Is there a partition of U into disjoint sets $U_1, U_2, \ldots, U_K$ such that the sum of the sizes of the items in each U_i is B or less?

Let a set $A = \{a_1, \ldots, a_n\}$ of n elements, a size $s(a_i) \in Z^+$ for each $a_i \in A$, a positive integer bin capacity B and a positive integer K.

We construct the following graph which is a convex graph: Consider an independent set $S^i = \{s_1^i, s_2^i, \ldots, s^i_{s(a_i)}\}$ of $s(a_i)$ vertices and an independent set $T^i = t_1^i, t_2^i, \ldots, t^i_{s(a_i)-1}$ of $s(a_i)-1$ vertices for every $a_i \in A$, $1 \le i \le n$. We connect every $t_j^i \in T^i$ to vertices $s_j^i \in S^i$ and $s_{j+1}^i \in S^i$, $1 \le j \le s(a_i)-1$; let P_i, $1 \le i \le n$ be the resulting disconnected graphs, each containing $2s(a_i)-1$ vertices. Thus, we can associate P_i with $a_i \in A$, for every $i \in [1, n]$. We add an independent set $C = \{c_1, c_2, \ldots, c_{n-K}\}$ of $n-K$ vertices and we connect each c_j, $1 \le j \le n-K$ to every vertex of all sets S^i, $1 \le i \le n$; let G be the resulting graph. The graph G is a connected graph and it is illustrated in Fig. 3.2.

One can easily verify that the graph G is a convex graph; we define the sets X and Y as follows:

$$\begin{aligned} X &= \{s_1^1, s_2^1, \ldots, s^1_{s(a_1)}, s_1^2, s_2^2, \ldots, s^2_{s(a_2)}, \ldots, s_1^n, s_2^n, \ldots, s^n_{s(a_n)}\} \\ Y &= \{t_1^1, t_2^1, \ldots, t^1_{s(a_1)-1}, t_1^2, t_2^2, \ldots, t^2_{s(a_2)-1}, \ldots, t_1^n, t_2^n, \ldots, t^n_{s(a_n)-1}, c_1, c_2, \ldots c_{n-K}\} \end{aligned}$$

Since X is ordered so that for each element y in the vertex set Y the elements of X connected to y form an interval of X, the constructed bipartite graph $G = (X, Y; E)$ of Fig. 3.2 is convex on the vertex set X.

We now claim that the graph G has a k-path partition into K paths of length at most $k = 2B-2$ if and only if A can be partitioned into K disjoint sets $A_1, A_2, \ldots, A_K$ such that the sum of the sizes of the items in each A_i is B or less.

($\Longleftarrow$) Suppose now there exists a partition of A in $A_1, \ldots, A_K$ such that the sum of the sizes of the items in each A_i is B or less. We show how to find a k-path partition of G into K paths of length at most $k = 2B-2$. Let α_i be the number of items contained in each A_i, $1 \le i \le K$. We construct n paths of length $2s(a_j)-2$, $1 \le j \le n$, that is, the paths $p_j = [s_1^j, t_1^j, s_2^j, t_2^j, s_3^j, \ldots, s^j_{s(a_j)-1}, t^j_{s(a_j)-1}, s^j_{s(a_j)}]$, $1 \le j \le n$. Note that each path p_j corresponds to each subgraph P_j of G. Then, we use $\alpha_i - 1$ vertices of the set C to connect the α_i paths corresponding to the elements of the set A_i into one path of length $\alpha_i - 2 - \alpha_i + 2\sum_{a \in A_i} s(a) \le 2B-2$.

($\Longrightarrow$) We next suppose that G has a $(2B-2)$-path partition into K paths. Since the set X contains $\sum_{i=1}^n s(a_i)$ vertices and the set Y contains $\sum_{i=1}^n s(a_i) - K$ vertices, then a minimum path partition cannot contain less than K paths. Moreover, since each vertex $t_j^i \in T$ $(1 \le i \le n,\ 1 \le j \le s(a_i)-1)$

sees only the vertices s^i_j and s^i_{j+1} of X, a path containing vertices of the subgraph P_i can be connected to a path containing vertices of the subgraph $P_{i'}$ only through a vertex of the set C, which contains $n-K$ vertices. We claim that, in order to obtain a path partition of no more than K paths, we first have to construct n paths $p_i = [s^i_1, t^i_1, s^i_2, t^i_2, s^i_3, \ldots, s^i_{s(a_i)-1}, t^i_{s(a_i)-1}, s^i_{s(a_i)}]$, $1 \le i \le n$, and then we have to connect them using vertices of C in such a way that no path contains more than $2B-1$ vertices; note that both endpoints of each path p_i are in X and each p_i corresponds to a subgraph P_i. Indeed, let q_i be a subpath of p_i and let p_j be the $n-1$ paths corresponding to the $n-1$ subgraphs P_j, where $p_j = [s^j_1, t^j_1, s^j_2, t^j_2, s^j_3, \ldots, s^j_{s(a_j)-1}, t^j_{s(a_j)-1}, s^j_{s(a_j)}]$, $1 \le j \le n$ and $i \ne j$. Then, there exist vertices of the subgraph P_i that are not included in the path q_i, which form a path q'_i. Thus, we have to connect $n+1$ paths using $n-K$ vertices of the set C, which results to $K+1$ paths, a contradiction. Consequently, in order to obtain a path partition of no more than K paths, we first have to construct n paths p_i, $1 \le i \le n$, corresponding to the subgraphs P_i, and then we have to connect them using vertices of C in such a way that no path contains more than $2B-1$ vertices. Let $P' = \{p'_1, p'_2, \ldots, p'_K\}$ be the set of the paths of the $(2B-2)$-path partition of G. Each one of these K paths contains at most B vertices of X and if a vertex s^i_ℓ, $\ell \in [1, s(a_i)]$ belongs to a certain path then all vertices s^i_j, $1 \le j \le s(a_i)$, belong to the same path. Consequently, the set A can be partitioned into K disjoint sets $A_1, A_2, \ldots, A_K$ such that the sum of the sizes of the items in each A_i is B or less.

The theorem follows from the strong NP-completeness of Bin-Packing, since the transformation can be done easily in polynomial time. ▮

3.3 Quasi-Threshold Graphs

A graph G is called *quasi-threshold*, or QT-graph for short, if G contains no induced subgraph isomorphic to P_4 or C_4 (cordless path or cycle on 4 vertices); for definition and optimization problems on this class see [40, 55, 66, 72, 73]. The class of quasi-threshold graphs is a subclass of the class of cographs and contains the class of threshold graphs [18, 40]; see Fig. 3.1.

3.3.1 Structural properties

Let G be a QT-graph with vertex set $V(G)$ and edge set $E(G)$. The following lemma follows immediately from the fact that for every subset $S \subset V(G)$ and for a vertex $x \in S$, we have $N_{G[S]}[x] = N[x] \cap S$ and that $G - S$ is an induced subgraph.

Lemma 3.1. *[55, 73]: If G is a QT-graph, then for every subset $S \subset V(G)$, both $G[S]$ and $G[V(G) - S]$ are also QT-graphs.*

The following theorem provides important properties for the class of QT-graphs. For convenience, we define

$$cent(G) = \{x \in V(G) \mid N[x] = V(G)\}.$$

Theorem 3.2. *[55, 73]: The following three statements hold.*

(i) A graph G is a QT-graph if and only if every connected induced subgraph $G[S], S \subseteq V(G)$, satisfies $cent(G[S]) \ne \emptyset$.

(ii) A graph G is a QT-graph if and only if $G[V(G) - cent(G)]$ is a QT-graph.

(iii) Let G be a connected QT-graph. If $V(G) - cent(G[S]) \ne \emptyset$, then $G[V(G) - cent(G)]$ contains at least two connected components.

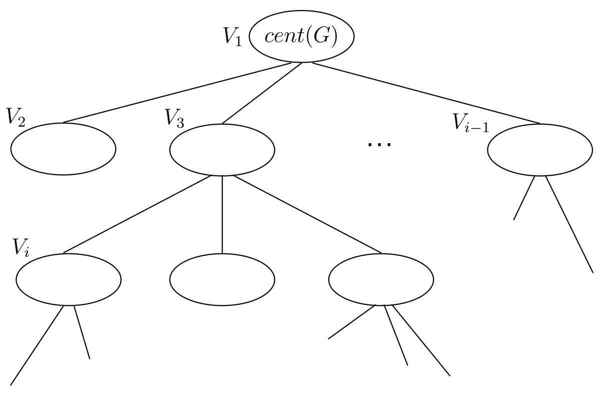


Figure 3.3: The typical structure of the cent-tree $T_c(G)$ of a QT-graph.

Let G be a connected QT-graph. Then $V_1 = cent(G)$ is not an empty set by Theorem 3.2. Let $G_1 = G$, and $G[V(G)-V_1] = G_2 \cup G_3 \cup \cdots \cup G_r$, where each G_i is a connected component of $G[V(G)-V_1]$ and $r \geq 3$. Then since each G_i is an induced subgraph of G, G_i is also a QT-graph, and so let $V_i = cent(G_i) \neq \emptyset$ for $2 \leq i \leq r$. Since each connected component of $G_i[V(G_i) - cent(G_i)]$ is also a QT-graph, we can continue this procedure until we get an empty graph. Then we finally obtain the following partition of $V(G)$:

$$V(G) = V_1 + V_2 + \cdots + V_k, \quad \text{where } V_i = cent(G_i).$$

Moreover we can define a partial order $\preceq$ on the set $\{V_1, V_2, \ldots, V_k\}$ as follows:

$$V_i \preceq V_j \ \text{ if } \ V_i = cent(G_i) \ \text{ and } \ V_j \subseteq V(G_i).$$

It is easy to see that the above partition of the vertex set $V(G)$ of the QT-graph G possesses the following properties.

Theorem 3.3. *[55, 73]: Let G be a connected QT-graph, and let $V(G) = V_1 + V_2 + \cdots + V_k$ be the partition defined above; in particular, $V_1 := cent(G)$. Then this partition and the partially ordered set $(\{V_i\}, \preceq)$ have the following properties:*

(P1) If $V_i \preceq V_j$, then every vertex of V_i and every vertex of V_j are joined by an edge of G.

(P2) For every V_j, $cent(G[\{\bigcup V_i \mid V_i \preceq V_j\}]) = V_j$.

(P3) For every two V_s and V_t such that $V_s \preceq V_t, G[\{\bigcup V_i \mid V_s \preceq V_i \preceq V_t\}]$ is a complete graph. Moreover, for every maximal element V_t of $(\{V_i\}, \preceq)$, $G[\{\bigcup V_i \mid V_1 \preceq V_i \preceq V_t\}]$ is a maximal complete subgraph of G.

The results of Theorem 3.3 provide structural properties for the class of QT-graphs. We shall refer to the structure that meets the properties of Theorem 3.3 as *cent-tree* of the graph G and denote it by $T_c(G)$. The cent-tree $T_c(G)$ (see Fig. 3.3) of a QT-graph is a rooted tree; it has nodes $V_1, V_2, \ldots, V_k$, root $V_1 := cent(G)$, and every node V_i is either a leaf or has at least two children. Moreover, $V_s \preceq V_t$ if and only if V_s is an ancestor of V_t in $T_c(G)$. Thus, we can state the following result.

Corollary 3.1. *A graph G is a QT-graph if and only if G has a cent-tree $T_c(G)$.*

Observation 3.1. Let G be a QT-graph and let $V = V_1 + V_2 + \cdots + V_k$ be the above partition of $V(G)$; $V_1 := cent(G)$. Let $S = \{v_s, v_{s+1}, \ldots, v_t, \ldots, v_q\}$ be a stable set such that $v_t \in V_t$ and V_t is a maximal element of $(V_i, \preceq)$ or, equivalently, V_t is a leaf node of $T_c(G)$, $s \leq t \leq q$. It is easy to see that S has the maximum cardinality $\alpha(G)$ among all the stable sets of G. On the other hand, the sets $\{\bigcup V_i \mid V_1 \preceq V_i \preceq V_t\}$, for every maximal element V_t of $(V_i, \preceq)$, provide a clique cover of size $\kappa(G)$ which is the smallest possible clique cover of G; that is $\alpha(G) = \kappa(G)$. Based on the Theorem 3.3 or, equivalently, on the properties of the cent-tree of G, it is easy to show that the clique number $\omega(G)$ equals the chromatic number $\chi(G)$ of the graph G; that is, $\chi(G) = \omega(G)$.

3.3.2 NP-completeness results

In order to prove that the k-path partition problem is NP-complete for quasi-threshold graphs, we use the 3-PARTITION problem. The formulation of the 3-PARTITION problem ([SP15] in [36]) is presented below.

3-PARTITION
Instance: Set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$, such that $\frac{1}{4}B < s(a) < \frac{1}{2}B$, and such that $\sum_{a \in A} s(a) = mB$.
Question: Can A be partitioned into m disjoined sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly three elements from A)?

Theorem 3.4. *The k-path partition problem is NP-complete for quasi-threshold graphs.*

Proof. The k-path partition problem is obviously in NP. In order to prove NP-hardness, we use a transformation from 3-PARTITION.

Let a set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ elements, a positive integer B and let positive integer sizes $s(a_i)$ for each $a_i \in A$ be given, such that $\frac{1}{4}B < s(a_i) < \frac{1}{2}B$, and such that $\sum_{a_i \in A} s(a_i) = mB$, $1 \leq i \leq 3m$. We may suppose that, for each $a_i \in A$, $s(a_i) > m$ (if not, then we can multiply all $s(a_i)$ and B with $m + 1$).

We construct the following graph which is a quasi-threshold graph: Consider a graph $G(V \cup C, E)$ having a clique $K_{a_i}(V_{a_i}, E_{a_i})$ on $s(a_i)$ vertices for each $a_i \in A$ such that $V_{a_i} \cap V_{a_j} = \emptyset$, $i \neq j$, and $V = \bigcup_{a_i \in A} V_{a_i}$. There are no edges in G between vertices in different cliques. In addition, G has $2m$ "connector" vertices $C = \{v_1, v_2, \ldots, v_{2m}\}$ which form a clique in G. Every $v_i \in C$ is connected to every $u \in V$. It is clear that G is a quasi-threshold graph.

We now claim that A has a 3-PARTITION, that is, A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that $\sum_{a \in A_i} s(a) = B$ for $1 \leq i \leq m$, if and only if G has a partition into m paths of length $k = B + 2$. Notice that the constraints on the item sizes ensure that each S_i must have exactly three elements from A.

($\Longrightarrow$) If A has a 3-PARTITION $A_i = \{x_i, y_i, z_i\}$, $1 \leq i \leq m$, then we can use the two elements $v_{2i-1}, v_{2i} \in C$ to connect the corresponding subgraphs K_{x_i}, K_{y_i} and K_{z_i} into a path $V_{x_i}, v_{2i-1}, V_{y_i}, v_{2i}, V_{z_i}$ of length $B + 2$.

($\Longleftarrow$) We next suppose that G has a $(B + 2)$-path partition into m paths, $P_1, P_2, \ldots, P_m$. Since G has $m(B + 2)$ vertices, each P_i must contain exactly $B + 2$ vertices. Because of the size constraints, each P_i must contain at least two connector vertices from C.

We claim that, in order to obtain a path partition of no more than m paths, we first have to construct $3m$ paths $p_1, p_2, \ldots, p_{3m}$ corresponding to the $3m$ cliques, and then we have to connect them using vertices of C in such a way that each path P_i, $1 \leq i \leq m$, contains exactly $B+2$ vertices. Indeed, let q_k be a subpath of path p_k corresponding to clique K_{a_k} and let p_j be the $3m-1$ paths corresponding to the rest $3m-1$ cliques. Then, there exist vertices of clique K_{a_k} that are not included in the path q_k, which form a path q'_k. Thus, we have to connect $3m+1$ paths using $2m$ vertices of the set C, which results to $m+1$ paths, a contradiction. Consequently, in order to obtain a path partition of m paths, we first have to construct $3m$ paths p_i, $1 \leq i \leq 3m$, corresponding to the cliques K_{a_i}, and then we have to connect them using vertices of C in such a way that each path contains exactly $B+2$ vertices.

Since we have $3m$ paths, corresponding to $3m$ cliques, and $2m$ connectors, each P_i must contain exactly two connector vertices. We claim that none of the paths P_i contains an edge between two vertices of clique C. Indeed, let P_k be a path containing an edge from clique C, that is, it contains two vertices of C. Since $s(a_i) < \frac{B}{2}$, $1 \leq i \leq 3m$, if P_k contains paths from two cliques, then its length is less than $B+2$. Thus, at least one more connector vertex from C is needed in order to connect at least one more path p_j to the path P_k. Consequently, we have a path, that is, P_k, using at least three connector vertices of C, a contradiction. Therefore, none of the paths P_i contains an edge between two vertices of clique C.

Since each P_i must contain exactly two connector vertices, no path P_i can have vertices from more than three cliques K_{a_i}. Since the length of each P_i is $B+2$, each P_i must cover the vertices of exactly three cliques K_{a_i} and the sizes of the corresponding three elements of A must add up to B. Consequently, the set A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that the sum of the sizes of the items in each A_i is equal to B.

The theorem follows from the strong NP-completeness of 3-PARTITION, since the transformation can be done easily in polynomial time. ∎

Since the class of quasi-threshold graphs is a subclass of interval graphs, which is a subclass of chordal graphs, the proof of the NP-hardness of the k-path partition problem for quasi-threshold graphs also establishes the NP-hardness of this problem for the class of interval and chordal graphs. Thus, we can state the following result.

Corollary 3.2. *The k-path partition problem is NP-complete for interval and chordal graphs.*

3.4 Concluding Remarks

We have studied the complexity of the k-path partition problem and proved that it is NP-complete for the class of convex and quasi-threshold graphs. Given that this problem is polynomially solvable for bipartite permutation graphs, we have sharpened the demarcation line between polynomially solvable and NP-hard cases of the k-path partition problem. The status of the problem remains open for the class of biconvex graphs; this class properly contains bipartite permutation graphs and is a proper subclass of convex graphs.

Chapter 4

NP-completeness Results for Coloring Problems on Subclasses of Chordal and Bipartite Graphs

4.1 Introduction

A *harmonious coloring* of a simple graph G is a proper vertex coloring such that each pair of colors appears together on at most one edge, while a *pair-complete coloring* of G is a proper vertex coloring such that each pair of colors appears together on at least one edge; the *harmonious chromatic number* $h(G)$ of the graph G is the least integer k for which G admits a harmonious coloring with k colors and its *achromatic number* $\psi(G)$ is the largest integer k for which G admits a pair-complete coloring with k colors.

The harmonious coloring problem is NP-complete on general graphs [47], while the pair-complete coloring problem was proved to be NP-hard on arbitrary graphs by Yannakakis and Gavril [93]. The formulations of the harmonious coloring problem and the pair-complete coloring problem in [15] are equivalent to the following formulations.

Harmonious Coloring Problem
Instance: Graph $G = (V, E)$, positive integer $K \leq |V|$.
Question: Is there a positive integer $k \leq K$ and a proper coloring using k colors such that each pair of colors appears together on at most one edge?

Pair-complete Coloring Problem
Instance: Graph $G = (V, E)$, positive integer $K \leq |V|$.
Question: Is there a positive integer $k \geq K$ and a proper coloring using k colors such that each pair of colors appears together on at least one edge?

The complexity of both problems has been extensively studied on various classes of perfect graphs such as cographs, interval graphs, bipartite graphs and trees [13, 40]; see Fig. 4.1 for their complexity status[1]. Bodlaender [11] provides a proof for the NP-completeness of the pair-complete coloring problem for disconnected cographs and disconnected interval graphs, and extends his results for the connected cases. His proof also establishes the NP-hardness of the harmonious coloring problem for disconnected interval graphs and disconnected cographs. It is worth noting that the problem of determining the harmonious chromatic number of a connected cograph is trivial, since in such a graph each vertex must receive a distinct color as it is at distance at most 2 from all other vertices [15]. Bodlaender's results establish the NP-hardness of the pair-complete coloring problem for the class of permutation graphs and, also, the NP-hardness of the harmonious coloring problem when restricted to disconnected permutation graphs. Extending the above results, we show that the harmonious coloring problem remains NP-complete on connected interval and permutation graphs.

Concerning the class of bipartite graphs and subclasses of this class (see Fig. 4.1), Farber et al. [32] show that the harmonious coloring problem and the pair-compete coloring problem are NP-complete for the class of bipartite graphs. In addition, Edwards et al. [30, 31] show that these problems are NP-complete for trees. Their results also establish the NP-completeness of these problems for the classes of convex graphs and disconnected bipartite permutation graphs. However, the complexity of these problems for connected bipartite permutation graphs and biconvex graphs is not straightforward.

Motivated by this issue we prove that the harmonious coloring problem and the pair-complete coloring problem is NP-complete for connected bipartite permutation graphs, and thus, the same holds for the class of biconvex graphs. Moreover, based on Bodlaender's results [11], we show that the pair-complete coloring problem is NP-complete for quasi-threshold graphs and that the harmonious coloring problem is NP-complete for disconnected quasi-threshold graphs. It has been shown that the harmonious coloring problem is polynomially solvable on threshold graphs. In this chapter we show that the pair-complete coloring problem is also polynomially solvable on this class by proposing a simple linear-time algorithm.

This chapter is organized as follows. In Section 4.2 we show that the harmonious coloring problem is NP-complete on connected interval and permutation graphs while in Section 4.3 we show that the problems are NP-complete on bipartite permutation graphs. In Section 4.4 we present structural properties of the class of quasi-threshold graphs and NP-completeness results on this class, while in Section 4.5 we show that the harmonious coloring problem is NP-complete on split graphs. In Section 4.6 we describe a simple linear-time algorithm for the pair-complete coloring problem on threshold graphs, and, finally, Section 4.7 concludes the chapter and discusses open problems.

[1]Figure 4.1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status for the harmonious coloring problem, the pair-complete coloring problem, and the k-path partition problem on these classes; for definitions of the classes shown, see [13, 40].

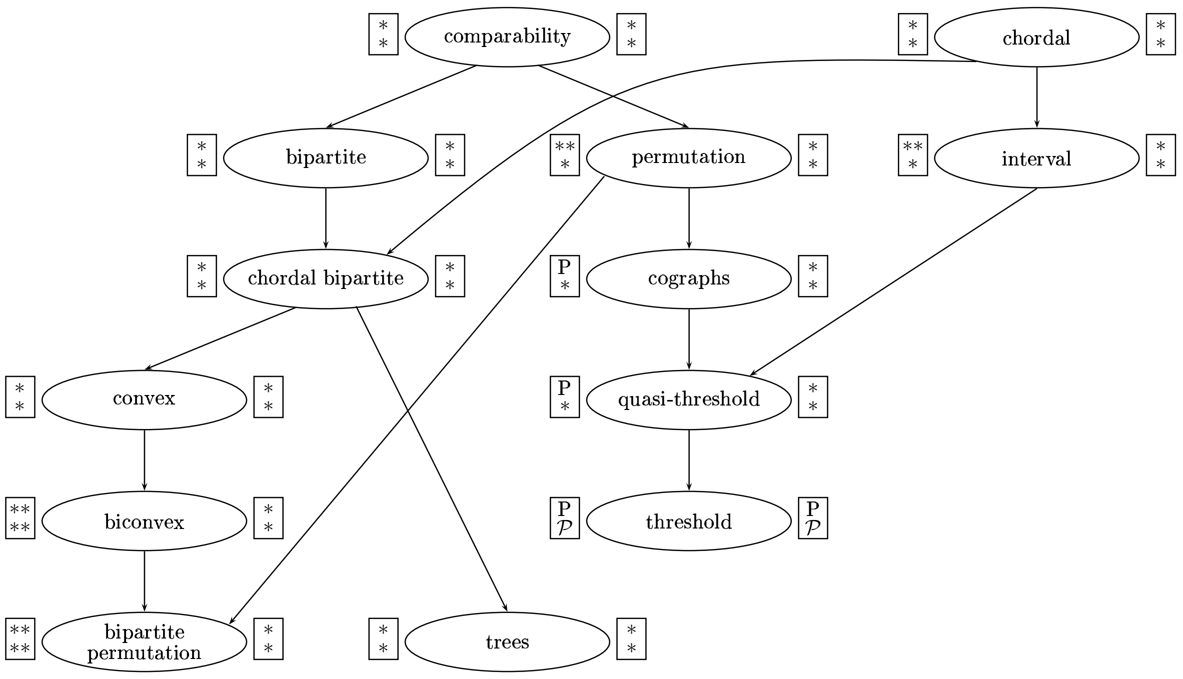


Figure 4.1: The complexity status of the coloring problems for some graph subclasses of comparability and chordal graphs. $A \rightarrow B$ indicates that class A contains class B. The box to the left (resp. right) of each class contains the status of the harmonious coloring (top), pair-complete coloring (bottom) problems on connected (resp. disconnected) graphs. ($*$): NP-complete, previously known; ($**$): NP-complete, new result; (P): polynomial, previously known; ($\mathcal{P}$): polynomial, new result.

4.2 Connected Interval and Permutation graphs

We next prove that the harmonious coloring problem is NP-complete for connected interval graphs; a graph G is an *interval graph* if its vertices can be put in one-to-one correspondence with a family of intervals on the real line such that two vertices are adjacent in G if and only if their corresponding intervals intersect.

Theorem 4.1. *Harmonious coloring is NP-complete when restricted to connected interval graphs.*

Proof. Harmonious coloring is obviously in NP. In order to prove NP-hardness, we use a transformation from a strongly NP-complete problem, that is, the 3-PARTITION problem. The formulation of the 3-PARTITION problem [36] is presented in Section 3.3.

Let a set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ elements, a positive integer B and let positive integer sizes $s(a_i)$ for each $a_i \in A$ be given, such that $\frac{1}{4}B < s(a_i) < \frac{1}{2}B$, and such that $\sum_{a_i \in A} s(a_i) = mB$. We may suppose that, for each $a_i \in A$, $s(a_i) > m$ (if not, then we can multiply all $s(a_i)$ and B with $m + 1$).

Extending the result of Bodlaender [11], we construct the following connected graph which is an interval and a permutation graph: Consider a clique with m vertices, a clique with B vertices, and add a vertex v that is connected to every vertex in the two cliques; let G_1 be the resulting graph. Next we construct for every $a_i \in A$ a tree T_i of depth one with $s(a_i)$ leaves and root x_i, that is, every leaf is adjacent to the root; note that there are $3m$ such trees $T_1, T_2, \ldots, T_{3m}$. Then we construct a path

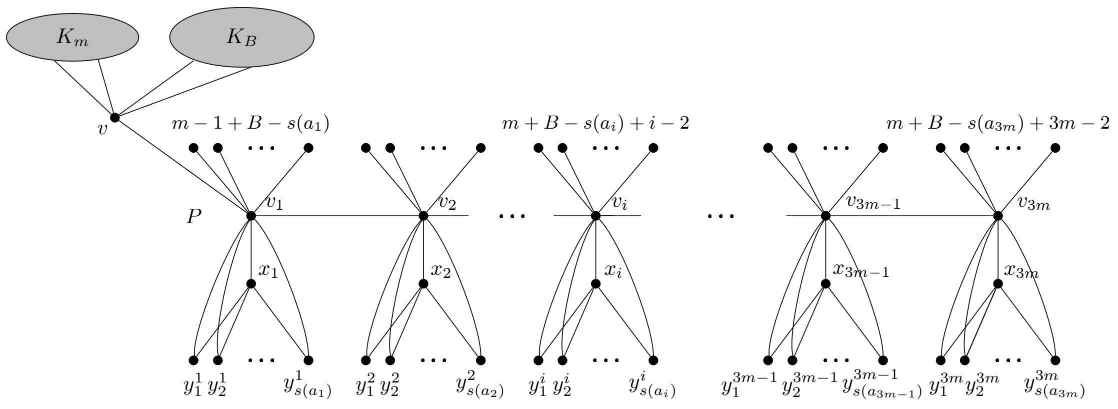


Figure 4.2: Illustrating the constructed connected interval and permutation graph G.

$P = [v_1, v_2, \ldots, v_{3m}]$ of $3m$ vertices, and we connect each vertex v_i of the path P to all the vertices of the tree T_i, $1 \le i \le 3m$. Additionally, for each vertex $v_i \in P$, we add $m - 1 + B - s(a_i) + i - 1$ vertices and connect them to vertex v_i; let G_2 be the resulting graph. Note that the graph $G_1 \cup G_2$ is disconnected. Finally, we add an edge to the graph $G_1 \cup G_2$ connecting vertices v_1 and v and let G be the resulting graph. The graph G is a connected graph and it is illustrated in Fig. 4.2.

One can easily verify that G is an interval graph. A clique can be represented as a number of intervals that share at least one point in common. Two cliques sharing a vertex u can be represented as a number of intervals such that one of them, which corresponds to u, shares at least one point with the intervals corresponding to the vertices of each clique. It is easy to see that the vertices of G can be put in one-to-one correspondence with a family of intervals on the real line such that two vertices are adjacent in G if and only if their corresponding intervals intersect.

It is easy to see that the total number of edges in G is

$$\binom{m}{2} + \binom{B}{2} + m + B + 3m + mB + 3m + mB + 3m(m-2) + 2mB + \sum_{i=1}^{3m} i = \binom{4m+B+1}{2}$$

For every harmonious coloring of G and every pair of distinct colors i, j, $i \neq j$, there must be at most one edge with its endpoints colored with i and j. Thus, it follows that the harmonious chromatic number cannot be less than $4m + B + 1$, and if it is equal to $4m + B + 1$ then we have, for every pair of distinct colors i, j, $1 \le i, j \le 4m + B + 1$, a unique edge with its end-points colored with i and j. Thus, we have an exact coloring of G; an *exact coloring* of G with k colors is a harmonious coloring of G with k colors in which, for each pair of colors i, j, there is exactly one edge (a, b) such that a has color i and b has color j.

We now claim that the harmonious chromatic number of G is (less or equal to) $4m + B + 1$ if and only if A can be partitioned in m sets $A_1, \ldots, A_m$ such that $\sum_{a \in A_j} s(a) = B$, for all j, $1 \le j \le m$.

($\Longleftarrow$) Suppose now a 3-partition of A in $A_1, \ldots, A_m$ such that $\forall j : \sum_{a \in A_j} s(a) = B$ exists. We show how to find a harmonious coloring of G using $4m + B + 1$ colors. We color the vertices of the first clique with colors $1, 2, \ldots, m$, the vertices of the second clique with $m + 1, m + 2, \ldots, m + B$, and vertex v with $m + B + 1$. For convenience and ease of presentation, let $\mathcal{M}$ be the set containing colors $1, 2, \ldots, m$, let $\mathcal{B}$ be the set containing colors $m + 1, m + 2, \ldots, m + B$, and let $\mathcal{K}$ be the set containing colors $m + B + 2, m + B + 3, \ldots, 4m + B + 1$. If $a_i \in A_j$ then we color the vertex x_i with color j. Each color $j \in \mathcal{M}$ is assigned to the three vertices corresponding to three a_i that have together exactly B neighbors of degree 2. We assign to each one of these B neighbors a different color from $\mathcal{B}$, and next we assign to each vertex v_i of the path P a distinct color from $\mathcal{K}$. Recall that each vertex v_i, $1 < i < 3m$, is connected to two other vertices of P, i.e., v_{i-1} and v_{i+1}, and $m + B + i - 1$ more vertices, vertex v_1 is connected to v_2, v and $m + B$ other vertices, while vertex v_{3m} is connected to v_{3m-1} and $m + B + 3m - 1$ more vertices (see Fig. 4.2).

Next, we color the rest $m - 1 + B - s(a_i) + i - 1$ neighbors of each v_i. We assign a distinct color from the set $\mathcal{M}\backslash c_i$ to $m - 1$ neighbors of v_i, where c_i is the color previously assigned to the vertex x_i. We next assign a distinct color from the set $\mathcal{B}\backslash C_i$ to $B - s(a_i)$ neighbors of v_i, where C_i is the set of the colors previously assigned to $s(a_i)$ neighbors of the vertex x_i. Finally, we assign a different color to the rest $i - 1$ neighbors of v_i, $3 \leq i \leq 3m$, using color $m + b + 1$ and the colors assigned to the vertices v_j, $1 \leq j \leq i - 2$. Note that, in order to color the $m + B - s(a_2)$ neighbors of v_2, we only need to use color $m + B + 1$ and colors from $\mathcal{M}$ and $\mathcal{B}$, while for the $m - 1 + B - s(a_1)$ neighbors of v_1 we only use colors from $\mathcal{M}$ and $\mathcal{B}$. A harmonious coloring of G using $4m + B + 1$ colors results, and thus, the harmonious chromatic number of G is $4m + B + 1$.

($\Longrightarrow$) We next suppose that the harmonious chromatic number of G is (less or equal to) $4m + B + 1$. Consider a harmonious coloring of G using $4m + B + 1$ colors. Without loss of generality we may suppose that the m vertices of the first clique have distinct colors from $\mathcal{M}$, while the B vertices of the second clique have distinct colors from $\mathcal{B}$. Also, without loss of generality, we color vertex v with color $m + B + 1$ since v is adjacent to all the vertices of the two cliques. Since v_{3m} is the vertex having the maximum degree, that is, $4m + B$, it has to take a color from $\mathcal{K}$. Indeed, if it takes a color from $\mathcal{M}$, then none of its neighbors can take a color from $\mathcal{M}$ and we cannot color $4m + B$ vertices using only $4m + B + 1 - m$ colors. Using similar arguments, we cannot color vertex v_{3m} using a color from $\mathcal{B}$ or the color $m + B + 1$. Thus, without loss of generality, we assign to v_{3m} the color $4m + B + 1$. We color all its neighbors with distinct colors from $\mathcal{M} \cup \mathcal{B} \cup \{m + B + 1\} \cup \mathcal{K}\backslash\{4m + B + 1\}$. Note that, vertex v_{3m-1} takes a color from $\mathcal{K}\backslash\{4m + B + 1\}$; let $4m + B$ be this color. Indeed, using similar arguments, it cannot take a color from $\mathcal{M} \cup \mathcal{B} \cup \{m + B + 1\} \cup \{4m + B + 1\}$. Note that, color $4m + B + 1$ cannot be assigned to any other vertex of G since any pair of colors $(4m + B + 1, j)$, $1 \leq j \leq 4m + B$, already appears in the harmonious coloring. Recall that, for every pair of distinct colors i, j, $1 \leq i, j \leq 4m + B + 1$, there is a unique edge with its end-points colored with i and j. Recursively, as can easily be proved by induction on i, the same holds for all $v_i \in P$, $1 \leq i \leq 3m - 2$, that is, v_i takes a color from $\mathcal{K}\backslash\mathcal{L}$, where $\mathcal{L}$ is the set containing colors $m + B + 1 + i + 1, m + B + 1 + i + 2, \ldots, 4m + B + 1$, which are the colors already assigned to vertices v_j, $i < j \leq 3m$.

Note that pairs (μ, ν), $\mu \in \mathcal{M}$, $\nu \in \mathcal{B}$, have not appeared yet. Since every pair of colors must appear, we assign these pairs to the mB edges that have both endpoints uncolored. Note that these edges are the edges (x_i, y^i_j), $1 \leq i \leq 3m$, $1 \leq j \leq s(a_i)$, where x_i corresponds to a_i and y^i_j corresponds to the j-th neighbor of x_i having degree 2. The vertices x_i cannot take a color from $\mathcal{B}$, otherwise its $s(a_i) > m$ uncolored neighbors y^i_j cannot be colored with m colors from $\mathcal{M}$. Thus, vertices x_i are assigned a color from $\mathcal{M}$ and vertices y^i_j are assigned a color from $\mathcal{B}$ (recall that $\frac{B}{4} < s(a_i) < \frac{B}{2}$). Note that the only uncolored vertices are $m - 1 + B - s(a_i) + i - 1$ neighbors of each v_i, $1 \leq i \leq 3m$. In order to color $m - 1 + B - s(a_i)$ of the uncolored neighbors of v_i, we use distinct colors from $(\mathcal{M} \cup \mathcal{B})\backslash\mathcal{F}$, where $\mathcal{F}$ is the set containing all colors already assigned to the $s(a_i) + 1$ neighbors of v_i. In order to color the last $i - 1$ uncolored neighbors of v_i, $i > 1$, we can only use colors from $\mathcal{K}\backslash\mathcal{L}\backslash\{m + B + 1 + i, m + B + i\}$ because the only unused pairs are $(m + B + 1 + i, j)$, where $m + B + 1 \leq j \leq m + B + 1 + i - 2$.

Finally, let $a_i \in A_j$ if and only if the vertex x_i (with neighbors y^i_j) is colored with color $j \in \mathcal{M}$. We claim that for all j, $\sum_{a \in A_j} s(a) = B$. Indeed, each color j must be adjacent to some colors from $\mathcal{B}$, and each color from $\mathcal{B}$ is assigned to exactly one vertex which is adjacent to all x_i colored with j. Hence, a correct 3-partition exists.

The theorem follows from the strong NP-completeness of 3-PARTITION, since the transformation can be done easily in polynomial time. ∎

We can easily show that the interval graph G illustrated in Fig. 4.2 is also a permutation graph. The graph G is an interval graph if and only if it is a chordal graph and the graph $\overline{G}$ is a comparability graph [40]. Moreover, one can easily verify that G admits an acyclic transitive orientation and, thus, it is a comparability graph. Since G and $\overline{G}$ are comparability graphs, it follows that G is a permutation graph [40]. Consequently, we can state the following theorem.

Theorem 4.2. *Harmonious coloring is NP-complete when restricted to connected permutation graphs.*

4.3 Bipartite Permutation Graphs

We next prove that the harmonious coloring problem is NP-complete for connected bipartite permutation graphs. A bipartite graph $G = (X, Y; E)$ is a *bipartite permutation graph* if and only if it has a strong ordering of its vertices [85]; a *strong ordering* of the vertices of $G = (X, Y; E)$ is an ordering $\{x_1, x_2, \ldots, x_r\}$ of the vertices in X and an ordering $\{y_1, y_2, \ldots, y_s\}$ of the vertices in Y such that whenever $x_i y_\ell, x_j y_m \in E$ with $i < j$ and $\ell > m$ then we also have $x_i y_m, x_j y_\ell \in E$ [85].

Theorem 4.3. *The harmonious coloring problem is NP-complete when restricted to connected bipartite permutation graphs.*

Proof. Harmonious coloring is obviously in NP. In order to prove NP-hardness, we use a transformation from 3-PARTITION.

Let a set $A = \{a_1, \ldots, a_{3m}\}$ of $3m$ elements, a positive integer b and let positive integer sizes $s(a_i)$ for each $a_i \in A$ be given, such that $\frac{1}{4}b < s(a_i) < \frac{1}{2}b$, $\sum_{a_i \in A} s(a_i) = mb$, and $1 \le i \le 3m$. We may suppose that, for each $a_i \in A$, $s(a_i) > m$ (if not, then we can multiply all $s(a_i)$ and b with $m+1$).

We construct the following connected graph which is a bipartite permutation graph: Consider a set $M = \{m_1, m_2, \ldots, m_m\}$ of m vertices, a set $B = b_1, b_2, \ldots, b_b$ of b vertices, and add a vertex v that is connected to every vertex in the two sets. We add a set $M' = \{m'_1, m'_2, \ldots, m'_{m-1}\}$ of $m-1$ vertices and a set $B' = \{b'_2, b'_3, \ldots, b'_b\}$ of $b-1$ vertices. We connect M' and B' to the vertices of M and B as follows: we connect each vertex m'_i, $1 \le i \le m-1$, to the vertices $m_{i+1}, m_{i+2}, \ldots, m_m$, and each vertex b_i, $1 \le i \le b-1$, to the vertices $b'_{i+1}, b'_{i+2}, \ldots, b'_b$. Next we construct for every $a_i \in A$ a tree T_i of depth one with $s(a_i)$ leaves, namely $y^i_1, y^i_2, \ldots, y^i_{s(a_i)}$, and root x_i, that is, every leaf is adjacent to the root; note that there are $3m$ such trees $T_1, T_2, \ldots, T_{3m}$. Then we add a set $P = \{p_1, p_2, \ldots, p_{3m}\}$ of $3m$ vertices, and we connect each vertex p_i to the root x_i of the tree T_i, $1 \le i \le 3m$. We also connect p_i, $2 \le i \le 3m$, to the $s(a_{i-1})$ leaves of the tree T_{i-1}. The vertex p_1 is also connected to the vertices of M' and the vertex v. Additionally, for each vertex $p_i \in P$, $2 \le i \le 3m$, we add vertices v^i_j, $1 \le j \le m-1+b-s(a_{i-1})+1+3m-i$ and connect them to vertex p_i. We also add vertices v^1_j, $1 \le j \le b+3m-1$ and connect them to the vertex p_1; let G be the resulting graph. The graph G is a connected graph and it is illustrated in Fig. 4.3.

One can easily verify that the graph G is a bipartite graph; let X and Y be its two stable sets. It is easy to show that the graph $G = (X, Y; E)$ admits a strong ordering of its vertices, and, thus, it is a bipartite permutation graph. Let $\mathcal{X}$ and $\mathcal{Y}$ be the orderings of the vertices of X and Y, respectively. We define $\mathcal{X}$ and $\mathcal{Y}$ as follows:

$$\begin{aligned}
\mathcal{X} &= \{b'_2, b'_3, \ldots, b'_b, v, m'_1, m'_2, \ldots, m'_{m-1}, v^1_1, v^1_2, \ldots, v^1_{b+3m-1}, \mathcal{X}_1, \mathcal{X}_3, \mathcal{X}_5, \ldots, \mathcal{X}_{3m-2}, x_{3m}\} \\
\mathcal{Y} &= \{b_1, b_2, \ldots, b_b, m_1, m_2, \ldots, m_m, \mathcal{Y}_1, \mathcal{Y}_3, \mathcal{Y}_5, \ldots, \mathcal{Y}_{3m-2}, y^{3m}_1, y^{3m}_2, \ldots, y^{3m}_{s(a_{3m})}\}
\end{aligned}$$

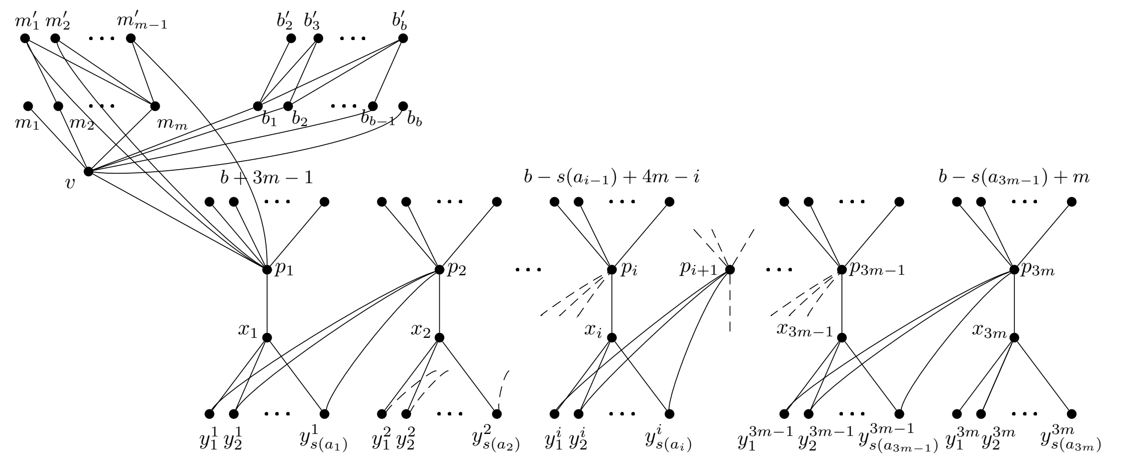


Figure 4.3: Illustrating the constructed connected bipartite permutation graph G.

where $\mathcal{X}_i = \{x_i, p_{i+1}, y_1^{i+1}, y_2^{i+1}, \ldots, y_{s(a_{i+1})}^{i+1}, v_1^{i+2}, v_2^{i+2}, \ldots, v_{4m+b-s(a_{i+1})-i-2}^{i+2}\}$, $i = 1, 3, 5, \ldots, 3m-2$, and $\mathcal{Y}_i = \{x_i, y_1^i, y_2^i, \ldots, y_{s(a_i)}^i, v_1^{i+1}, v_2^{i+1}, \ldots, v_{4m+b-s(a_i)-i-1}^{i+1}, x_{i+1}\}$, $i = 1, 3, 5, \ldots, 3m-2$.

It is easy to see that the total number of edges in G is

$$\binom{m}{2} + \binom{b}{2} + m + b + 3m^2 + 3mb + 3m + mb + \sum_{i=1}^{3m-1} i = \binom{4m+b+1}{2}$$

For every harmonious coloring of G and every pair of distinct colors i, j, $i \neq j$, there must be at most one edge with its endpoints colored with i and j. Thus, it follows that the harmonious chromatic number cannot be less than $4m+b+1$, and if it is equal to $4m+b+1$ then we have, for every pair of distinct colors i, j, $1 \leq i, j \leq 4m+b+1$, a unique edge with its end-points colored with i and j. Thus, we have an exact coloring of G; an *exact coloring* of G with k colors is a harmonious coloring of G with k colors in which, for each pair of colors i, j, there is exactly one edge ab such that a has color i and b has color j.

We now claim that the harmonious chromatic number of G is (less or equal to) $4m+b+1$ if and only if A can be partitioned in m sets $A_1, \ldots, A_m$ such that $\sum_{a \in A_j} s(a) = b$, for all j, $1 \leq j \leq m$.

($\Longleftarrow$) Suppose now a 3-partition of A in $A_1, \ldots, A_m$ such that $\forall j : \sum_{a \in A_j} s(a) = b$ exists. We show how to find a harmonious coloring of G using $4m+b+1$ colors. We color the vertices of the sets M and M' with colors $1, 2, \ldots, m$, the vertices of the sets B and B' with colors $m+1, m+2, \ldots, m+b$, and vertex v with $m+b+1$. For convenience and ease of presentation, let $\mathcal{M}$ be the set containing colors $1, 2, \ldots, m$, let $\mathcal{B}$ be the set containing colors $m+1, m+2, \ldots, m+b$, and let $\mathcal{K}$ be the set containing colors $m+b+2, m+b+3, \ldots, 4m+b+1$. If $a_i \in A_j$ then we color the vertex corresponding to a_i with color j. Each color $j \in \mathcal{M}$ is assigned to the three vertices x_i corresponding to three a_i that have together exactly b neighbors of degree 2. We assign to each one of these b neighbors a different color from $\mathcal{B}$, and next we assign to each vertex p_i of the set P a distinct color from $\mathcal{K}$. Recall that each vertex p_i, $1 \leq i \leq 3m$, is connected to $m+b+1+3m-i$ vertices (see Fig. 4.3).

Next, we color the rest $m-1+b-s(a_{i-1})+1+3m-i$ neighbors of each p_i, $1 < i \leq 3m$. We assign a distinct color from the set $\mathcal{M} \backslash \{c_i\}$ to $m-1$ neighbors of p_i, where c_i is the color previously assigned to the vertex x_i corresponding to a_i. We next assign a distinct color from the set $\mathcal{B} \backslash C_i$ to $b - s(a_{i-1})$ neighbors of p_i, where C_i is the set of the colors previously assigned to $s(a_{i-1})$ neighbors of the vertex

x_{i-1} corresponding to a_{i-1}. Finally, we assign a different color to the rest $1+3m-i$ neighbors of p_i, using color $m+b+1$ and the colors assigned to the vertices p_j, $i+1 \leq j \leq 3m$. Note that, we have assigned a color to m neighbors of p_1, and, thus, in order to color the rest $b+3m-1$ neighbors of p_1, we use colors from $\mathcal{K}$ and $\mathcal{B}$. A harmonious coloring of G using $4m+b+1$ colors results, and thus, the harmonious chromatic number of G is $4m+b+1$.

($\Longrightarrow$) We next suppose that the harmonious chromatic number of G is (less or equal to) $4m+b+1$. Consider a harmonious coloring of G using $4m+b+1$ colors. Without loss of generality we may suppose that the m vertices of the set M have distinct colors from $\mathcal{M}$, while the b vertices of the set B have distinct colors from $\mathcal{B}$. Also, without loss of generality, we color vertex v with color $m+b+1$, since v is adjacent to all the vertices of the two sets, and vertex p_1 with color $c_{p_1} = m+b+2$. Note that p_1 is the vertex having the maximum degree, that is, $4m+b$, and, thus, color $m+b+2$ is adjacent to all colors, because we color all uncolored neighbors of p_1 with distinct colors from $\mathcal{M} \cup \mathcal{B} \cup \mathcal{K} \backslash \{c_{p_1}\}$. We claim that every vertex p_i, $1 < i \leq 3m$, takes a color from $\mathcal{K}$. Indeed, let $c_m \in \mathcal{M}$ be a color assigned to p_2. The degree of vertex p_2 is equal to $4m+b-1$. However, color c_m can be adjacent to $(m-1+b+3m+1)-(1+1) < 4m+b-1$ other colors, and, thus, we need one more color in order to color one more neighbor of p_2. Using similar arguments, we show that vertex p_2 cannot take a color from $\mathcal{B} \cup \{m+b+1, m+b+2\}$, and thus it takes a color from $\mathcal{K} \backslash \{c_{p_1}\}$. Recursively, as can easily be proved by induction on i, the same holds for all $p_i \in P$, $2 < i \leq 3m$, that is, p_i takes a color from $\mathcal{K} \backslash \mathcal{L}$, where $\mathcal{L}$ is the set containing colors $c_{p_1}, c_{p_2}, \ldots, c_{p_{i-1}}$, which are the colors already assigned to vertices p_j, $1 \leq j < i$. Note that, if $c_{\mathcal{K}}$ is a color from $\mathcal{K} \cup \{m+b+1\}$, then it cannot be assigned to any other vertex of G since any pair of colors $(c_{\mathcal{K}}, j)$, $1 \leq j \leq 4m+b+1$, already appears in the harmonious coloring. Recall that, for every pair of distinct colors i, j, $1 \leq i, j \leq 4m+b+1$, there is a unique edge with its end-points colored with i and j.

We now show that all the vertices of the set B' receive colors from $\mathcal{B}$. Since each vertex $u_i \in B'$, $2 \leq i \leq b$, is adjacent to at least one vertex in B, none of them can take color $m+b+1$. Let $u \in B'$ be one vertex taking a color from $\mathcal{M}$, and let d_u be its degree, while all the other vertices take colors from $\mathcal{B}$. The number of edges of G having one endpoint colored with a color from $\mathcal{M}$ that have not appeared yet is $mb - d_u$. Also, the number of edges of G having one endpoint colored with a color from $\mathcal{B}$ that have not appeared yet is mb. Thus, the number of pais that have not appeared yet in G, is $mb - d_u + mb - mb = mb - d_u$, while the number of uncolored edges is mb, that is, the edges of the form $x_i y_j^i$, $1 \leq i \leq 3m$, $1 \leq j \leq s(a_i)$. This implies that we need more colors, and consequently, all the vertices of the set B' receive colors from $\mathcal{B}$. Using similar arguments we can show that the vertices of the set M' receive colors from $\mathcal{M}$.

Note that pairs (μ, ν), $\mu \in \mathcal{M}$, $\nu \in \mathcal{B}$, have not appeared yet. Since every pair of colors must appear, we assign these pairs to the mB edges that have both endpoints uncolored. Note that these edges are the edges $x_i y_j^i$, $1 \leq i \leq 3m$, $1 \leq j \leq s(a_i)$, where x_i corresponds to a_i and y_j^i corresponds to the j-th neighbor of x_i having degree 2. The vertices x_i cannot take a color from $\mathcal{B}$, otherwise the $s(a_i) > m$ uncolored neighbors y_j^i cannot be colored with m colors from $\mathcal{M}$. Thus, vertices x_i are assigned a color from $\mathcal{M}$ and vertices y_j^i are assigned a color from $\mathcal{B}$ (recall that $\frac{b}{4} < s(a_i) < \frac{b}{2}$). Note that it is easy to assign a distinct color to the $4m+b-s(a_{i-1})-i$ neighbors of each p_i, $1 < i \leq 3m$ that have degree equal to one; recall that $m-1$ neighbors of p_1 belonging to the set M' are already assigned a color from $\mathcal{M}$. If c_{p_i} is the color of vertex p_i, we use distinct colors from $\mathcal{M} \cup \mathcal{B} \cup \mathcal{K} \backslash \{c_{x_i}, \mathcal{F}, \mathcal{L}, c_{p_i}\}$, where $\mathcal{F}$ is the set containing all colors already assigned to the $s(a_{i-1})+1$ neighbors of p_i and $c_{x_i} \in \mathcal{M}$ is the color already assigned to vertex x_i.

Finally, let $a_i \in A_j$ if and only if the vertex x_i (with neighbors y_j^i) is colored with color $j \in \mathcal{M}$. We claim that for all j, $\sum_{a \in A_j} s(a) = b$. Indeed, each color j must be adjacent to some colors from $\mathcal{B}$, and each color from $\mathcal{B}$ is assigned to exactly one vertex which is adjacent to all x_i colored with j. Hence, a correct 3-partition exists.

The theorem follows from the strong NP-completeness of 3-PARTITION, since the transformation can be done easily in polynomial time. ∎

We have shown that the connected bipartite permutation graph G presented in this chapter, has $\binom{4m+b+1}{2}$ edges and $h(G) = 4m + b + 1$. In [30] it was shown that if G is a graph with exactly $\binom{k}{2}$ edges, then a proper vertex coloring of G with k colors is pair-complete if and only if it is a harmonious coloring. Thus, if G is a graph with $\binom{k}{2}$ edges, then $\psi(G) = k$ if and only if $h(G) = k$ [15]. Consequently, for the graph G, which is a bipartite permutation graph, we have that $\psi(G) = 4m + b + 1$ and, thus, our results also prove that the achromatic number is NP-complete for connected bipartite permutation graphs. Consequently, we can state the following theorem.

Theorem 4.4. *The pair-complete coloring problem is NP-complete when restricted to connected bipartite permutation graphs.*

We have shown that harmonious coloring and pair-complete coloring are NP-complete problems for the class of bipartite permutation graphs. Consequently, the two problems are NP-complete for the class of biconvex graphs, which properly contains bipartite permutation graphs. A bipartite graph $G = (X, Y; E)$ is *convex* on the vertex set X if X can be ordered so that for each element y in the vertex set Y the elements of X connected to y form an interval of X; G is *biconvex* if it is convex on both X and Y. Consequently, we can state the following result.

Corollary 4.1. *The harmonious coloring problem and the pair-complete coloring problem are NP-complete for biconvex graphs.*

4.4 Quasi-Threshold Graphs

A graph G is called *quasi-threshold*, or *QT*-graph for short, if G contains no induced subgraph isomorphic to P_4 or C_4 (cordless path or cycle on 4 vertices). Structural properties of the class of quasi-threshold graphs are presented in Section 3.3.

4.4.1 NP-completeness results

In order to prove the NP-completeness of the pair-complete coloring problem for cographs and interval graphs, Bodlaender [11] constructs an instance of a disconnected graph which is simultaneously a cograph and an interval graph and modifies it in order to obtain a connected instance of a graph which remains a cograph and an interval graph. One can easily verify that the constructed graphs are also quasi-threshold graphs. Thus, his proof also establishes the NP-hardness of the pair-complete coloring problem for the class of quasi-threshold graphs, as well as the NP-hardness of the harmonious coloring problem for disconnected quasi-threshold graphs. Consequently, we state the following result.

Corollary 4.2. *The pair-complete coloring problem is NP-complete for quasi-threshold graphs; the harmonious coloring problem is NP-complete for disconnected quasi-threshold graphs.*

4.5 Split Graphs

We next show that the harmonious coloring problem is NP-complete for split graphs, by exhibiting a reduction from the chromatic number problem for general graphs, which is known to be NP-complete [36].

Let G be an arbitrary graph with n vertices $v_1, v_2, \ldots, v_n$ and m edges $e_1, e_2, \ldots, e_m$. We construct in polynomial time a split graph $\widehat{G}$, where $V(\widehat{G}) = K + I$, as follows: the independent set I consists of n vertices $\widehat{v}_1, \widehat{v}_2, \ldots, \widehat{v}_n$ which correspond to the vertices $v_1, v_2, \ldots, v_n$ of the graph G and the clique K consists of m vertices $\widehat{u}_1, \widehat{u}_2, \ldots, \widehat{u}_m$ which correspond to the edges $e_1, e_2, \ldots, e_m$ of G. A vertex $\widehat{u}_t \in K$, $1 \leq t \leq m$, is connected to two vertices $\widehat{v}_i, \widehat{v}_j \in I$, $1 \leq i, j \leq n$, if and only if the corresponding vertices v_i and v_j are adjacent in G. Note that, every $\widehat{u}_i \in K$ sees all the vertices of the clique K and two vertices of the independent set I; thus, $|E(\widehat{G})| = \frac{m(m-1)}{2} + 2m$.

We claim that the graph G has a chromatic number $\chi(G)$ if and only if the split graph $\widehat{G}$ has a harmonious chromatic number $h(\widehat{G}) = \chi(G) + m$.

Let $c_i \in \{1, \ldots, \chi(G)\}$ be the color assigned to the vertex $v_i \in G$, $1 \leq i \leq n$, in a minimum coloring of G. We assign the color c_i to the vertex $\widehat{v}_i$ of the set I and a distinct color from the set $\{\chi(G) + 1, \ldots, \chi(G) + m\}$ to each vertex of the clique K. Since two adjacent vertices of G receive a different color, the neighbors of each $\widehat{u}_i \in K$ belonging to the independent set have distinct colors. Moreover, every vertex $\widehat{v}_i \in I$ sees $|N_G(v_i)|$ vertices of the clique K, where $N_G(v_i)$ is the neighborhood of the vertex v_i in G. Thus, every pair of colors appears in at most one edge. In addition, the number of colors assigned to the set I is equal to $\chi(G)$ and the number of colors assigned to the clique is equal to m. This results to a harmonious coloring of $\widehat{G}$ using $\chi(G) + m$ colors, which is minimum since the vertices of the set I cannot receive a color assigned to a vertex of the clique K.

Conversely, a harmonious coloring of $\widehat{G}$ using $h(\widehat{G}) = \chi(G) + m$ colors assigns m colors to the vertices of the clique K and $\chi(G)$ colors to the vertices of the set I. Note that, $\chi(G)$ is the minimum number of colors so that vertices $\widehat{v}_i, \widehat{v}_j$ having a neighbor in common are assigned different colors. Since v_i, v_j are adjacent in G, it follows that we have a minimum coloring of G using $\chi(G)$ colors.

Thus, we have proved the following result.

Theorem 4.5. *Harmonious coloring is NP-complete for split graphs.*

4.6 Threshold Graphs

In this section we study the pair-complete coloring problem on threshold graphs and describe a linear-time algorithm based on structural properties of the class of threshold graphs.

The concept of threshold graph was introduced by Chvátal and Hammer in 1977 [17]. A graph G is a *threshold graph* [17, 18, 40] if and only if G does not contain $2K_2$, P_4 or C_4 as induced subgraphs. There exists an alternative equivalent definition [67]: A graph is threshold if there exists a partition of $V(G)$ into disjoint sets K, I and an ordering $\{u_l, u_2, \ldots, u_n\}$ of the nodes in I such that K induces a clique in G, I is a stable set of vertices and $N_G(u_1) \subseteq N_G(u_2) \subseteq \cdots \subseteq N_G(u_n)$. A partition of $V(G)$ satisfying the above definition will be called a (K, I) *partition* of G.

4.6.1 A tree structure

The class of threshold graphs is a subclass of quasi-threshold graphs; see Fig. 4.1. Consequently, for a threshold graph G there is a tree structure which meets the properties of G, that is, the cent-tree $T_c(G)$ which is similar to the cent-tree of a QT-graph; see Fig. 3.3. Since a threshold graph G does not contain

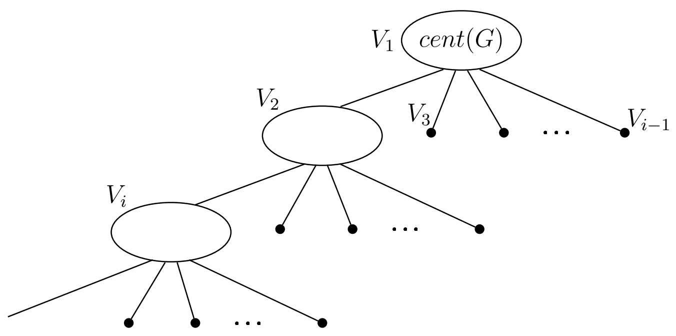


Figure 4.4: The typical structure of the cent-tree $T_c(G)$ of a threshold graph.

an induced subgraph isomorphic to $2K_2$, each non-leaf vertex V_i has $k_i \geq 2$ children, where at most one of them is a non-leaf child while the rest $k_i - 1$ children are leaves containing only one vertex; see Fig. 4.4. Note that the cent-tree $T_c(G)$ of a threshold graph G represents a (K, I) *partition* of G; equivalently, given a (K, I) *partition* of G, we can construct the cent-tree $T_c(G)$.

4.6.2 Pair-complete coloring problem: a polynomial solution

The pair-complete coloring problem on a threshold graph G can be solved in linear time using its cent-tree $T_c(G)$; see Fig. 4.4. The vertices V_i of the leftmost path of the tree form a clique and thus each vertex $v_i \in V(G)$ belonging to this path must receive a distinct color. If n' is the number of the vertices of G that belong to the leftmost path of $T_c(G)$, then we claim that the vertices of G take colors from the set $C = \{1, 2, \ldots, n'\}$ and the achromatic number $\psi(G)$ is $\psi(G) = n'$. Indeed, let $C' \subset C$ be the set of the colors assigned to the leftmost leaf of $T_c(G)$ and let $c'_i \in C'$. If we assign a new color, say, $n' + 1$, to an uncolored vertex of $T_c(G)$ then the pair $(n'+1, c'_i)$ cannot appear, which is a contradiction. Consequently, we use the set C to assign colors to the uncolored leaves of $T_c(G)$ in such a way that no vertex $v_i \in V(G)$ takes a color already assigned to an ancestor that belongs to the leftmost path.

Note that, if n' is the number of the vertices of G that belong to the leftmost path of $T_c(G)$, then n' equals the clique number $\omega(G)$, and, thus, $\psi(G) = \omega(G)$. Furthermore, based on the properties of the cent-tree $T_c(G)$, it is easy to show that the clique number equals the chromatic number $\chi(G)$ of the graph G; that is, $\chi(G) = \omega(G)$. Thus, we propose a linear-time algorithm which holds for connected and disconnected threshold graphs.

It is worth noting that a disconnected threshold graph includes only one connected component having more than one vertex; each one of the rest of the connected components consists of only one vertex; otherwise there would exist a subgraph isomorphic to $2K_2$. Consequently, we can color the isolated vertices using one color we have already used. Thus, the fourth step of the algorithm is performed when the graph is disconnected. In conclusion, we state the following theorem:

Theorem 4.6. *Let G be a threshold graph. The pair-complete coloring problem is solved in linear time on G and the achromatic number is $\psi(G) = \omega(G)$.*

4.7 Concluding Remarks

We have studied the complexity of the harmonious coloring problem and the pair-complete coloring problem on subclasses of bipartite graphs. Specifically, we have proved that both problems are NP-

Algorithm Pair_Complete_Coloring

Input: a threshold graph G;

Output: a pair-complete coloring of G having $\psi(G) = \omega(G)$;

```
1. Construct the cent-tree $T_c(G)$ of $G$;
2. Color the vertices of the leftmost path (clique) of $T_c(G)$ with distinct colors from the set $C = \{1, 2, \ldots, \psi(G)\}$.
3. Color each leaf vertex of $T_c(G)$ using a color already assigned to the sibling vertex that belongs to the leftmost path of $T_c(G)$ and contains a clique.
4. If there are any isolated vertices, color them using a color from the set $C$.
```

Algorithm 2: Algorithm Pair_Complete_Coloring

complete for the class of connected bipartite permutation graphs and, thus, they are NP-complete for the class of biconvex graphs. Apart from the NP-completeness results, we have proposed a linear-time algorithm for the pair-complete coloring problem on a subclass of chordal graphs namely threshold graphs. Furthermore, we have shown that the connected interval graph G presented in Fig 4.2, which is also a permutation graph, has $\binom{4m+B+1}{2}$ edges and $h(G) = 4m + B + 1$. In [30] it was shown that if G is a graph with exactly $\binom{k}{2}$ edges, then a proper vertex coloring of G with k colors is pair-complete if and only if it is a harmonious coloring. Thus, if G is a graph with $\binom{k}{2}$ edges, then $\psi(G) = k$ if and only if $h(G) = k$ [15]. Consequently, for the graph G, which is simultaneously an interval and a permutation graph, we have that $\psi(G) = 4m + B + 1$ and, thus, our results could be also used to prove that the achromatic number is NP-complete for connected interval and permutation graphs.

Chapter 5

A Linear-time Algorithm for the k-fixed-endpoint Path Cover Problem on Cographs

5.1 Introduction

In this chapter, we study a variant of the path cover problem, namely, the *k-fixed-endpoint path cover problem*, or kPC for short. Given a graph G and a subset $\mathcal{T}$ of k vertices of $V(G)$, a k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the k vertices of $\mathcal{T}$ are all endpoints of the paths in $\mathcal{P}$. Note that, if $\mathcal{T}$ is empty, that is, $k = 0$, the stated problem coincides with the classical path cover problem. We show that the k-fixed-endpoint path cover problem can be solved in linear time on the class of cographs. The proposed algorithm is simple, requires linear space, and also enables us to solve some path cover related problems, such as the 1HP and 2HP, on cographs within the same time and space complexity. We next define the kPC problem.

Problem kPC. Let G be a graph and let $\mathcal{T}$ be a set of k vertices of $V(G)$. A *k-fixed-endpoint path cover* of the graph G with respect to $\mathcal{T}$ is a path cover of G such that all vertices in $\mathcal{T}$ are endpoints of paths in the path cover; a *minimum k-fixed-endpoint path cover* of G with respect to $\mathcal{T}$ is a k-fixed-endpoint path cover of G with minimum cardinality; the *k-fixed-endpoint path cover problem* (kPC) is to find a minimum k-fixed-endpoint path cover of the graph G.

We show that the k-fixed-endpoint path cover problem (kPC) has a polynomial-time solution in the class of complement reducible graphs, or cographs [24, 59]. The class of cographs is defined recursively as follows: (i) a single vertex graph is a cograph; (ii) if G is a cograph then its complement $\overline{G}$ is also a cograph; (iii) if G_1 and G_2 are cographs satisfying $V(G_1) \cap V(G_2) = \emptyset$, then their union $G_1 \cup G_2$ is also a cograph. Thus, cographs are formed from a single vertex under the closure of the operations of union and complement. Cographs were independently discovered under various names and were shown to have the following two remarkable properties: they are P_4 restricted graphs and they have a unique tree representation (see [59]). This tree, called the *co-tree*, forms the basis for fast algorithms for problems such as isomorphism, coloring, clique detection, clusters, minimum weight dominating sets [22, 23], and also for the path cover problem [61, 70].

Contribution. In this chapter, we study the complexity status of the k-fixed-endpoint path cover problem (kPC) on the class of cographs, and show that this problem can be solved in polynomial time when the input is a cograph. More precisely, we establish a lower bound on the size of a minimum k-fixed-endpoint path cover of a cograph G on n vertices and m edges. We then define path operations, and prove structural properties for the paths of such a path cover, which enable us to describe a simple algorithm for the kPC problem. The proposed algorithm runs in time linear in the size of the input graph G, that is, in $O(n+m)$ time, and requires linear space. To the best of our knowledge, this is the first linear-time algorithm for solving the kPC problem on the class of cographs.

The proposed algorithm for the kPC problem can also be used to solve the 1HP and 2HP problems on cographs within the same time and space complexity. Moreover, we have designed our algorithm so that it produces a minimum k-fixed-endpoint path cover of a cograph G that contains a large number of paths with both their endpoints in $\mathcal{T}$ (we can easily find a graph G and a set $\mathcal{T}$ of k vertices of $V(G)$ so that G admits two minimum k-fixed-endpoint path covers with different numbers of paths having both their endpoints in $\mathcal{T}$; for example, consider the graph G with vertex set $V(G) = \{a, b, c, d\}$, edge set $E(G) = \{ab,\ bc,\ ac,\ cd\}$, and $\mathcal{T} = \{a, b\}$). Thus, we can also use our algorithm to solve problem A on cographs within the same time and space complexity.

Related Work. The class of cographs has been extensively studied and several sequential and/or parallel algorithms for recognition and for classical combinatorial optimization problems have been proposed. Corneil et al. [24] proposed a linear-time recognition algorithm for cographs. Jung [54] studied the existence of a Hamiltonian path or cycle in a cograph, while Lin et al. [61] proposed an optimal algorithm for the path cover problem on cographs. Nakano et al. [70] proposed an optimal parallel algorithm which finds and reports all the paths in a minimum path cover of a cograph in $O(\log n)$ time using $O(n/\log n)$ processors on a PRAM model. Furthermore, quite recently Nikolopoulos [72] solved the Hamiltonian problem on quasi-threshold graphs (a subclass of cographs) in $O(\log n)$ time using $O(n+m)$ processors on a PRAM model. Sequential algorithms for optimization problems on other related classes of graphs (superclasses of cographs) have been also proposed: Giakoumakis et al. [38] solved the recognition problem and also the problems of finding the clique number, the stability number and the chromatic number for P_4-sparse graphs [45] (a proper superclass of cographs) in linear sequential time. Hochstättler and Tinhofer [46] presented a sequential algorithm for the path cover problem on this class of graphs, which runs in $f(n)+O(n)$ time, where $f(n)$ is the time complexity for the construction of a tree representation of a P_4-sparse graph. Also, Giakoumakis et al. [38] studied hamiltonicity properties for the class of P_4-*tidy* graphs (a proper superclass of P_4-sparse graphs); see also [13]. Recently, Hsieh et al. [49] presented an $O(n+m)$-time sequential algorithm for the Hamiltonian problem on a distance-hereditary graph and also proposed a parallel implementation of their algorithm which solves the problem in $O(\log n)$ time using $O((n+m)/\log n)$ processors on a PRAM model. A unified approach to solving the Hamiltonian problems on distance-hereditary graphs was presented in [50], while Hsieh [48] presented an efficient parallel strategy for the 2HP problem on the same class of graphs. Algorithms for the path cover problem on other classes of graphs were proposed in [4, 51, 86].

Road Map. The chapter is organized as follows. In Section 5.2 we establish the notation and related terminology, and we present background results. In Section 5.3 we describe our linear-time algorithm for the kPC problem, while in Section 5.4 we prove its correctness and compute its time and space complexity. Finally, in Section 5.5 we conclude the chapter and discuss possible future extensions.

5.2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$.

5.2.1 The co-tree

The cographs admit a tree representation unique up to isomorphism. Specifically, we can associate with every cograph G a unique rooted tree $T_{co}(G)$ called the co-tree (or, modular decomposition tree [68]) of G having the following properties:

1. Every internal node of $T_{co}(G)$ has at least two children;

2. The internal nodes of $T_{co}(G)$ are labelled by either P (P-node) or S (S-node) in such a way that the labels alternate along every path in $T_{co}(G)$ starting at the root;

3. Each leaf of $T_{co}(G)$ corresponds to a vertex in V such that $(x, y) \in E$ if and only if the lowest common ancestor of the leaves corresponding to x and y is an S-node.

It is shown that for every cograph G the co-tree $T_{co}(G)$ is unique up to isomorphism and it can be constructed sequentially in linear time [22, 24].

For convenience and ease of presentation, we binarize the co-tree $T_{co}(G)$ in such a way that each of its internal nodes has exactly two children [61, 70]. Let t be an internal node of $T_{co}(G)$ with children $t_1, t_2, \ldots, t_k$ where $k \geq 3$. We replace node t by $k-1$ nodes $t'_1, t'_2, \ldots, t'_{k-1}$ such that t'_1 has children t_1 and t_2 and each t'_i $(2 \leq i < k)$ has children t'_{i-1} and t_{i+1}. We shall refer to the binarized version of $T_{co}(G)$ as the modified co-tree of G and will denote it by $T(G)$. Thus, the left and right child of an internal node t of $T(G)$ will be denoted by t_ℓ and t_r, respectively.

Let t be an internal node of $T(G)$. Then $G[t]$ is the subgraph of G induced by the subset V_t of the vertex set $V(G)$, which contains all the vertices of G that have as common ancestor in $T(G)$ the node t. For simplicity, we will denote by V_ℓ and V_r the vertex sets $V(G[t_\ell])$ and $V(G[t_r])$, respectively.

5.2.2 Cographs and the kPC problem

Let G be a cograph, $\mathcal{T}$ be a set of k vertices of $V(G)$, and let $\mathcal{P}_\mathcal{T}(G)$ be a minimum k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ of size $\lambda_\mathcal{T}$; note that the size of $\mathcal{P}_\mathcal{T}(G)$ is the number of paths it contains. The vertices of the set $\mathcal{T}$ are called *terminal* vertices, and the set $\mathcal{T}$ is called the *terminal set* of G, while those of $V(G) - \mathcal{T}$ are called *non-terminal or free* vertices. Thus, the set $\mathcal{P}_\mathcal{T}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices, that is, $u, v \in \mathcal{T}$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}$ and the other in $V(G)-\mathcal{T}$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G)-\mathcal{T}$; if P_f consists of only one vertex, say, u, then $u \in V(G)-\mathcal{T}$.

Note that all the internal vertices of the paths of $\mathcal{P}_\mathcal{T}(G)$ are free vertices. Moreover, a semi-terminal path may consist of only one vertex which is a terminal vertex, while a terminal path contains at least two vertices. The set of the non-terminal paths in a minimum kPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Thus, we have

$$\lambda_\mathcal{T} = |N| + |S| + |T| \tag{5.1}$$

From the definition of the k-fixed-endpoint path cover problem (kPC), we can easily conclude that the number of paths in a minimum kPC can not be less than the number of terminal vertices divided by two. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T|$. Thus, the following proposition holds:

Proposition 5.1. *Let G be a cograph and let $\mathcal{T}$ be a terminal set of G. Then $|\mathcal{T}| = |S| + 2|T|$ and $\lambda_\mathcal{T} \geq \lceil \frac{|\mathcal{T}|}{2} \rceil$.*

Clearly, the size of a kPC of a cograph G, as well as the size of a minimum kPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_\mathcal{T} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G; hereafter, $F(V) = F(V(G))$. Then we have the following proposition:

Proposition 5.2. *Let G be a cograph and let $\mathcal{T}$ be a terminal set of G. If $\lambda_\mathcal{T}$ is the size of a minimum kPC of G, then $\lambda_\mathcal{T} \leq |F(V)| + |\mathcal{T}|$.*

Let t be an internal node of the tree $T(G)$. Then $\lambda_\mathcal{T}(t)$ denotes the number of paths in a minimum kPC of the graph $G[t]$, and let t_ℓ and t_r be the left and the right child of node t, respectively. We denote by $\mathcal{T}_\ell$ and $\mathcal{T}_r$ the terminal vertices in V_ℓ and V_r, respectively, where $V_\ell = V(G[t_\ell])$ and $V_r = V(G[t_r])$. Let N_ℓ, S_ℓ and T_ℓ be the sets of the non-terminal, semi-terminal and terminal paths in a minimum kPC of $G[t_\ell]$, respectively. Similarly, let N_r, S_r and T_r be the sets of the non-terminal, semi-terminal and terminal paths in a minimum kPC of $G[t_r]$, respectively. Obviously, Eq. (5.1) holds for $G[t]$ as well, with t being either an S-node or a P-node, that is,

$$\lambda_\mathcal{T}(t) = |N_t| + |S_t| + |T_t| \tag{5.2}$$

where N_t, S_t and T_t are the sets of the non-terminal, the semi-terminal and the terminal paths in a minimum kPC of $G[t]$, respectively. If t is a P-node, then a minimum kPC $\mathcal{P}_\mathcal{T}(t)$ of $G[t]$ is $\mathcal{P}_\mathcal{T}(t) = \mathcal{P}_\mathcal{T}(t_\ell) \cup \mathcal{P}_\mathcal{T}(t_r)$, where $\mathcal{P}_\mathcal{T}(t_\ell)$ and $\mathcal{P}_\mathcal{T}(t_r)$ are minimum kPCs corresponding to $G[t_\ell]$ and $G[t_r]$, respectively, and $\lambda_\mathcal{T}(t) = \lambda_\mathcal{T}(t_\ell) + \lambda_\mathcal{T}(t_r)$. Furthermore, for the case of a P-node we have

$$\begin{aligned} |N_t| &= |N_\ell| + |N_r| \\ |S_t| &= |S_\ell| + |S_r| \\ |T_t| &= |T_\ell| + |T_r| \end{aligned}$$

Thus, we focus on computing a minimum kPC of the graph $G[t]$ for the case where t is an S-node.

Before describing our algorithm, we establish a lower bound on the size $\lambda_\mathcal{T}(t)$ of a minimum kPC $\mathcal{P}_\mathcal{T}(t)$ of a graph $G[t]$. More precisely, we prove the following lemma.

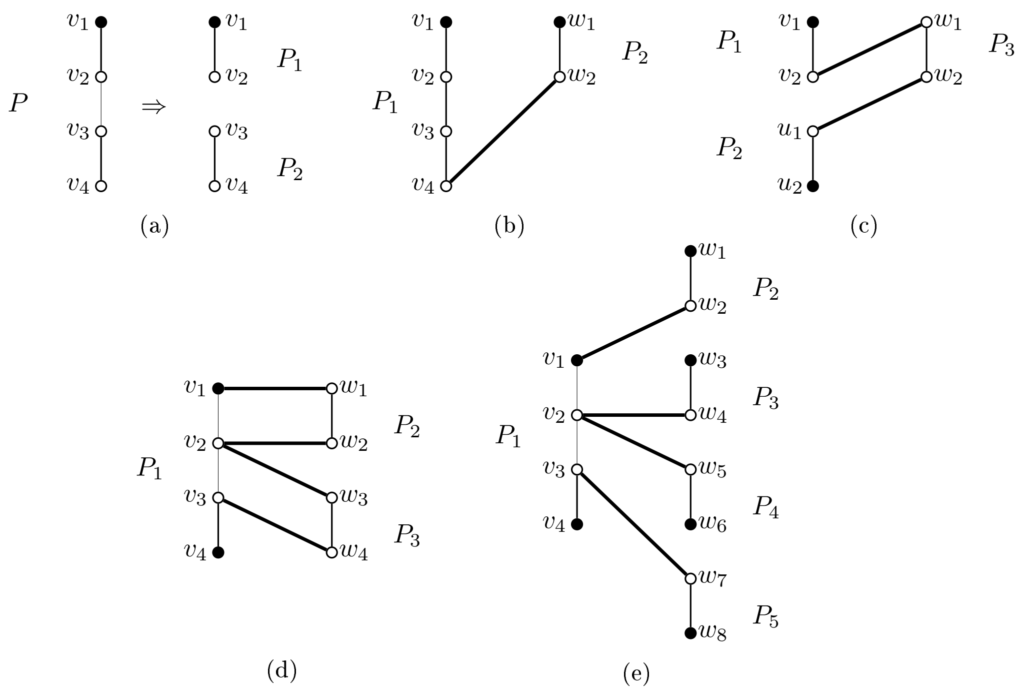


Figure 5.1: Illustrating (a) break, (b) connect, (c) bridge, (d) insert, and (e) connect-bridge operations.

Lemma 5.1. *Let t be an internal node of $T(G)$ and let $\mathcal{P}_\mathcal{T}(t_\ell)$ and $\mathcal{P}_\mathcal{T}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. Then $\lambda_\mathcal{T}(t) \geq \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil,\ \lambda_\mathcal{T}(t_\ell) - |F(V_r)|,\ \lambda_\mathcal{T}(t_r) - |F(V_\ell)|\}$.*

Proof. Clearly, according to Proposition 5.1 and since $G[t]$ is a cograph, we have $\lambda_\mathcal{T}(t) \geq \lceil\frac{|\mathcal{T}_t|}{2}\rceil$. We will prove that $\lambda_\mathcal{T}(t) \geq \lambda_\mathcal{T}(t_\ell) - |F(V_r)|$. Assume that $\lambda_\mathcal{T}(t) < \lambda_\mathcal{T}(t_\ell) - |F(V_r)|$. Consider removing from this path cover all the vertices in V_r. What results is a set of paths which is clearly a kPC for $G[t_\ell]$. Since the removal of a free vertex in $F(V_r)$ will increase the number of paths by at most one, we obtain a kPC of $G[t_\ell]$ of size at most $\lambda_\mathcal{T}(t) + |F(V_r)|$. The assumption $\lambda_\mathcal{T}(t) < \lambda_\mathcal{T}(t_\ell) - |F(V_r)|$ guarantees that $\lambda_\mathcal{T}(t) + |F(V_r)| < \lambda_\mathcal{T}(t_\ell)$, contradicting the minimality of $\mathcal{P}_\mathcal{T}(t_\ell)$. Using similar arguments we can show that $\lambda_\mathcal{T}(t) \geq \lambda_\mathcal{T}(t_r) - |F(V_\ell)|$. Hence, the lemma follows. ∎

We next define four operations on paths of a minimum kPC of the graphs $G[t_\ell]$ and $G[t_r]$, namely *break*, *connect*, *bridge* and *insert* operations; these operations are illustrated in Fig. 5.1.

- *Break* operation: Let $P = [p_1, p_2, \ldots, p_k]$ be a path of $\mathcal{P}_\mathcal{T}(t_r)$ or $\mathcal{P}_\mathcal{T}(t_\ell)$ of length k. We say that we *break* the path P in two paths, say, P_1 and P_2, if we delete an arbitrary edge of P, say the edge $p_i p_{i+1}$ $(1 \leq i < k)$, in order to obtain two paths which are $P_1 = [p_1, \ldots, p_i]$ and $P_2 = [p_{i+1}, \ldots, p_k]$. Note that we can break the path P in at most k trivial paths.

- *Connect* operation: Let $P_1 = [p_1, \ldots, p_1']$ be a non-terminal or a semi-terminal path of $\mathcal{P}_\mathcal{T}(t_\ell)$ (resp. $\mathcal{P}_\mathcal{T}(t_r)$) and let $P_2 = [p_2, \ldots, p_2']$ be a non-terminal or a semi-terminal path of $\mathcal{P}_\mathcal{T}(t_r)$ (resp. $\mathcal{P}_\mathcal{T}(t_\ell)$). We say that we *connect* the path P_1 with the path P_2, if we add an edge which joins two free endpoints of the two paths.

- *Bridge* operation: Let $P_1 = [p_1, \ldots, p'_1]$ and $P_2 = [p_2, \ldots, p'_2]$ be two paths of the set $N_\ell \cup S_\ell$ (resp. $N_r \cup S_r$) and let $P_3 = [p_3, \ldots, p'_3]$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *bridge* the two paths P_1 and P_2 using path P_3 if we connect the free endpoint of P_1 with one endpoint of P_3 and the free endpoint of P_2 with the other endpoint of P_3. The result is a path having both endpoints in $G[t_\ell]$ (resp. $G[t_r]$).

- *Insert* operation: Let $P_1 = [t_1, p_1, \ldots, p'_1, t'_1]$ be a terminal path of the set T_ℓ (resp. T_r) and let $P_2 = [p_2, \ldots, p'_2]$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *insert* the path P_2 into P_1, if we replace the first edge of P_1, that is, the edge t_1p_1, with the path $[t_1, p_2, \ldots, p'_2, p_1]$. Thus, the resulting path is $P_1 = [t_1, p_2, \ldots, p'_2, p_1, \ldots, p'_1, t'_1]$.
  Note that we can replace every edge of the terminal path so that we can insert at most $|F(\{P_1\})|+1$ non-terminal paths, where $F(\{P_1\})$ is the set of the free vertices belonging to the path P_1. If the terminal path $P_1 = [t_1, p_1, \ldots, p^\ell_1, p^r_1, \ldots, p'_1, t'_1]$ is constructed by connecting a semi-terminal path of S_ℓ, say, $P_\ell = [t_1, p_1, \ldots, p^\ell_1]$ with a semi-terminal path of S_r, say, $P_r = [p^r_1, \ldots, p'_1, t'_1]$, then it obviously has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. In this case, if $P_2 \in N_\ell$ (resp. N_r) we can only replace the edges of P_1 that belong to $G[t_r]$ (resp. $G[t_\ell]$). On the other hand, if P_2 has one endpoint, say, p_2, in N_ℓ and the other, say, p'_2, in N_r, we insert P_2 into P_1 as follows: $P_1 = [t_1, p_1, \ldots, p^\ell_1, p'_2, \ldots, p_2, p^r_1, \ldots, p'_1, t'_1]$.

We can combine the Connect and Bridge operations to perform a new operation on paths which we call a *connect-bridge* operation; such an operation is depicted in Fig. 5.1(e) and is defined below.

- *Connect-Bridge* operation: Let $P_1 = [t_1, p_1, \ldots, p_k, t'_1]$ be a terminal path of the set T_ℓ (resp. T_r) and let $P_2, P_3, \ldots, P_s$ be semi-terminal paths of the set S_r (resp. S_ℓ), where s is odd and $3 \leq s \leq 2k+3$. We say that we *connect-bridge* the paths $P_2, P_3, \ldots, P_s$ using vertices of P_1, if we perform the following operations:
  (i) connect the path P_2 with the path $[t_1]$;
  (ii) bridge $r = \frac{s-3}{2}$ pairs of semi-terminal paths using vertices $p_1, p_2, \ldots, p_r$;
  (iii) connect the path $[p_{r+1}, \ldots, p_k, t'_1]$ with the last semi-terminal path P_s.

We point out that the Connect-Bridge operation produces two paths having one endpoint in $G[t_\ell]$ (resp. $G[t_r]$) and the other endpoint in $G[t_r]$ (resp. $G[t_\ell]$) and $\frac{s-3}{2}$ paths having both endpoints in $G[t_r]$ (resp. $G[t_\ell]$).

5.3 The Algorithm

We next present an optimal algorithm for the kPC problem on cographs. Our algorithm takes as input a cograph G and a subset $\mathcal{T}$ of its vertices, and finds the paths of a minimum kPC of G in linear time; it works as follows:

where the description of the subroutine *process*( ) is as follows:

process (node t)

Input: node t of the modified co-tree $T(G)$ of the input graph G.

Output: a minimum kPC $\mathcal{P}_\mathcal{T}(t)$ of the cograph $G[t]$.

Algorithm Minimum_kPC

Input: a cograph G and a set of vertices $\mathcal{T}$;

Output: a minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of the cograph G;

1. Construct the co-tree $T_{co}(G)$ of G and make it binary; let $T(G)$ be the resulting tree;
2. Execute the subroutine *process*(*root*), where *root* is the root node of the tree $T(G)$; the minimum kPC $\mathcal{P}_{\mathcal{T}}(root) = \mathcal{P}_{\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

Algorithm 3: Algorithm Minimum_kPC_Cograph

```
1. if $t$ is a leaf
   then return($\{u\}$), where $u$ is the vertex associated with the leaf $t$;
   else {$t$ is an internal node that has a left and a right child denoted by $t_\ell$ and $t_r$, resp.}
        $\mathcal{P}_{\mathcal{T}}(t_\ell) \leftarrow process(t_\ell)$;
        $\mathcal{P}_{\mathcal{T}}(t_r) \leftarrow process(t_r)$;
2. if $t$ is a P-node
   then return($\mathcal{P}_{\mathcal{T}}(t_\ell) \cup \mathcal{P}_{\mathcal{T}}(t_r)$);
3. if $t$ is an S-node
   then if $|N_\ell| \leq |N_r|$ then swap($\mathcal{P}_{\mathcal{T}}(t_\ell), \mathcal{P}_{\mathcal{T}}(t_r)$);
        case 1:  $|S_\ell| \geq |S_r|$
             call $kPC\_1$;
        case 2:  $|S_\ell| < |S_r|$
             if  $|N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor \leq |F(S_\ell \cup N_\ell)|$
             then  call $kPC\_2\_a$;
             else  call $kPC\_2\_b$;
```

We next describe the subroutine *process*( ) in the case where t is an S-node of $T(G)$. Note that, if $|N_\ell| \leq |N_r|$, we swap $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ and thus we have $|N_\ell| \geq |N_r|$. Consequently, we distinguish the following two cases: (1) $|S_\ell| \geq |S_r|$, and (2) $|S_\ell| < |S_r|$.

Case 1: $|S_\ell| \geq |S_r|$

Let SN_r be the set of non-terminal paths obtained by breaking the set $S_r \cup N_r$ into $|N_\ell| - 1 + \lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor$ non-terminal paths; in the case where $|N_\ell| - 1 + \lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor \geq F(S_r \cup N_r)$, the paths of SN_r are trivial (recall that $F(S_r \cup N_r)$ is the set of free vertices belonging to the set $S_r \cup N_r$). The paths of SN_r are used to bridge at most $2\lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor$ semi-terminal paths of S_ℓ and, if $|SN_r| - \lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor > 0$, at most $|N_\ell|$ non-terminal paths of N_ℓ. Note that $|SN_r| \leq |F(S_r \cup N_r)|$. We can construct the paths of a kPC using the following procedure:

Procedure kPC_1

1. connect the $|S_r|$ paths of S_r with $|S_r|$ paths of S_ℓ;
2. bridge $2\lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor$ semi-terminal paths of S_ℓ using $\lfloor\frac{|S_\ell|-|S_r|}{2}\rfloor$ paths of SN_r;

3. bridge the non-terminal paths of N_ℓ using $|N_\ell| - 1$ non-terminal paths of SN_r; this produces non-terminal paths with both endpoints in $G[t_\ell]$, unless $N_\ell \leq |F(S_r \cup N_r)| - \lfloor \frac{|S_\ell|-|S_r|}{2} \rfloor$ where we obtain one non-terminal path with one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;

4. if $|N_\ell| \leq |F(S_r \cup N_r)| - \lfloor \frac{|S_\ell|-|S_r|}{2} \rfloor$ insert the non-terminal path obtained in Step 3 into one terminal path which is obtained in Step 1;

5. if $|T_r| = |S_\ell| = 0$ and $|F(S_r \cup N_r)| \geq |N_\ell|$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and insert it into a terminal path of T_ℓ;

6. if $|T_r| = |S_r| = 0$ and $|F(N_r)| \geq |N_\ell| + \lfloor \frac{|S_\ell|}{2} \rfloor$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and use it to connect two semi-terminal paths of S_ℓ;

7. if $|S_\ell| - |S_r|$ is odd and there is at least one free vertex in $S_r \cup N_r$ which is not used in Steps 1–4, or there is a non-terminal path having one endpoint in $G[t_\ell]$ and the other in $G[t_r]$, connect one non-terminal path with one semi-terminal path of S_ℓ;

8. connect-bridge the rest of the semi-terminal paths of S_ℓ (at most $2(|F(T_r)| + |T_r|)$) using vertices of T_r;

9. insert non-terminal paths obtained in Step 3 into the terminal paths of T_r;

Based on the procedure kPC_1, we can compute the cardinality of the sets N_t, S_t and T_t, and thus, since $\lambda'_{\mathcal{T}}(t) = |N_t| + |S_t| + |T_t|$, the number of paths in the kPC constructed by the procedure at node $t \in T(G)$. In this case, the values of $|N_t|$, $|S_t|$ and $|T_t|$ are the following:

$$\begin{array}{rcl} |N_t| & = & \max\{\mu - \alpha,\ 0\} \\ |S_t| & = & \min\{\sigma_\ell,\ \max\{\sigma_\ell - 2(|F(T_r)| + |T_r|),\ \delta(\sigma_\ell)\}\} \\ |T_t| & = & |S_r| + \min\{\lfloor \frac{|S_\ell|-|S_r|}{2} \rfloor,\ |F(S_r \cup N_r)|\} + |T_\ell| + |T_r| + \frac{\sigma_\ell - |S_t|}{2} \end{array} \tag{5.3}$$

where

$$\begin{array}{rcl} \sigma_\ell & = & |S_\ell| - |S_r| - 2\min\{\lfloor \dfrac{|S_\ell| - |S_r|}{2} \rfloor, |F(S_r \cup N_r)|\} \\ \mu & = & \max\{|N_\ell| - \pi_r,\ \max\{1 - |S_\ell|,\ 0\}\} - \min\{\max\{\min\{|N_\ell| - \pi_r,\ \delta(|S_\ell| - |S_r|)\},\ 0\}, \\ & & \max\{\min\{F(S_r \cup N_r) - \lfloor \dfrac{|S_\ell| - |S_r|}{2} \rfloor,\ 1\},\ 0\}\} - \dfrac{1}{2}\max\{2(|F(T_r)| + |T_r|) - \sigma_\ell,\ 0\} \\ \alpha & = & \min\{\max\{\min\{\pi_r - |N_\ell|,\ 1\},\ 0\},\ \max\{|T_\ell|,\ 0\}\} \end{array}$$

and

$$\pi_r = \max\{|F(S_r \cup N_r)| - \lceil \frac{|S_\ell| - |S_r|}{2} \rceil,\ 0\}.$$

In Eq. (5.3), σ_ℓ is the number of semi-terminal paths of S_ℓ that are not connected or bridged at Steps 1 and 2. Furthermore, π_r is the number of free vertices in the set $S_r \cup N_r$ that are not used to bridge semi-terminal paths of S_ℓ at Step 2 and δ is a function which is defined as follows: $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise. Note that at most $|F(T_r)| + |T_r|$ non-terminal paths can be inserted into the terminal paths of T_r or the terminal paths can connect-bridge at most $2(|F(T_r)| + |T_r|)$ semi-terminal paths.

Case 2: $|S_\ell| < |S_r|$

In this case, we need $|N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ paths of $G[t_\ell]$ in order to bridge $|N_r|$ non-terminal paths of N_r and $2\lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ semi-terminal paths of S_r. If $|N_\ell| < |N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ we break the non-terminal paths of N_ℓ into at most $|F(N_\ell)|$ paths; in the case where $|F(N_\ell)| < |N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ we also use (at most $|F(S_\ell)|$) vertices of S_ℓ. Let $p = \min\{|N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor,\ |F(S_\ell \cup N_\ell)|\}$. We distinguish two cases:

2.a $|N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor \leq |F(S_\ell \cup N_\ell)|$.

In this case, $p = |N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ and the number of non-terminal paths (or free vertices) of $G[t_\ell]$ is sufficient to bridge non-terminal paths of N_r and semi-terminal paths of S_r.

In detail, let SN_ℓ be the set of non-terminal paths obtained by breaking the set $S_\ell \cup N_\ell$ into p non-terminal paths. If $p < |N_\ell|$ then $SN_\ell = N_\ell$. The paths of SN_ℓ are used to bridge $2\lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ semi-terminal paths of S_r and all the non-terminal paths of N_r. Obviously, $|SN_\ell| \leq |F(S_\ell \cup N_\ell)|$. Note that, if $p < |N_\ell|$ then the non-terminal paths of N_r are used to bridge the paths of N_ℓ. More precisely, we use paths of the set SN_r (it is the set of non-terminal paths that we get by breaking the set $S_r \cup N_r$) in order to obtain $|N_\ell| - \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ non-terminal paths. If $p \geq |N_\ell|$ we set $SN_r = N_r$ and we use at most $|F(S_\ell \cup N_\ell)|$ paths obtained by $S_\ell \cup N_\ell$ in order to bridge non-terminal paths of N_r and semi-terminal paths of S_r, that is, we use the set SN_ℓ. As a result, we construct $\lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ terminal paths having both of their endpoints in $G[t_r]$ and we have at least one non-terminal path, if $p < |N_\ell|$, and exactly one non-terminal path, otherwise. Note that, in the second case, we can construct the non-terminal path in such a way that one endpoint is in SN_ℓ and the other is in N_r. Consequently, we can connect the path to a semi-terminal path of S_r, in the case where $|S_r| - |S_\ell|$ is an odd number, or we can insert it into a terminal path which is obtained by connecting $|S_\ell|$ paths of S_ℓ with $|S_\ell|$ paths of S_r. We construct the paths of a kPC at node $t \in T(G)$ using the following procedure:

Procedure kPC_2_a

1. connect the $|S_\ell|$ paths of S_ℓ with $|S_\ell|$ paths of S_r;
2. if $|T_\ell| = |T_r| = 0$ and $p \geq |N_\ell|$, use N_r to bridge $p - \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor + 1$ paths of SN_ℓ and use the constructed non-terminal path having both of its endpoints in $G[t_\ell]$ to bridge two semi-terminal paths of S_r;
3. bridge semi-terminal paths of S_r using paths of SN_ℓ;
4. if $|T_r| = 0, |T_\ell| \neq 0$, $p \geq |N_\ell|$ and $|F(S_r \cup N_r)| \geq |SN_\ell| - \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and use a terminal path of T_ℓ to insert the constructed non-terminal path;
5. bridge the remaining paths of SN_ℓ using the paths of SN_r. This produces non-terminal paths one of which has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;
6. if $|S_r| - |S_\ell|$ is odd, we connect one non-terminal path with one semi-terminal path of S_r;
7. insert at most $|F(T_r)| + |T_r|$ non-terminal paths obtained in Step 5 into the terminal paths of T_r;

Based on the path operations performed by procedure kPC_2_a, we can show that the sets N_t, S_t and T_t, have the following cardinalities:

$$\begin{aligned} |N_t| &= \max\{\mu - \alpha,\ 0\} \\ |S_t| &= \delta(|S_r| - |S_\ell|) \\ |T_t| &= |S_\ell| + \lfloor\tfrac{|S_r|-|S_\ell|}{2}\rfloor + |T_\ell| + |T_r| \end{aligned} \tag{5.4}$$

where

$$\begin{aligned}\mu &= \max\{|N_\ell| - F(S_r \cup N_r),\ 0\} - \lfloor\frac{|S_r| - |S_\ell|}{2}\rfloor - \delta(|S_r| - |S_\ell|) - |F(T_r)| - |T_r| \\ \alpha &= \min\{\max\{\min\{F(S_r \cup N_r) - |N_\ell|,\ 1\},\ 0\},\ \max\{|T_\ell|,\ 0\}\}\end{aligned}$$

Recall that $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise.

2.b $|N_r| + \lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor > |F(S_\ell \cup N_\ell)|$.

In this case, $p = |F(S_\ell \cup N_\ell)|$ and the number of non-terminal paths (or free vertices) of $G[t_\ell]$ (either in N_ℓ or in S_ℓ) is not sufficient to bridge non-terminal paths of N_r and semi-terminal paths of S_r.

In detail, let SN_ℓ be the set of the trivial, non-terminal paths, obtained by breaking the set $S_\ell \cup N_\ell$ into $|F(S_\ell \cup N_\ell)|$ non-terminal paths. Note that $SN_\ell = F(S_\ell \cup N_\ell)$. Similar to Case 1, we can construct the paths of a kPC at node $t \in T(G)$ using the following procedure:

Procedure kPC_2_b

1. connect the $|S_\ell|$ paths of S_ℓ with $|S_\ell|$ paths of S_r;
2. bridge $2\lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ semi-terminal paths of S_r using $\lfloor\frac{|S_r|-|S_\ell|}{2}\rfloor$ paths of SN_ℓ;
3. bridge the non-terminal paths of N_r using the rest of the non-terminal paths of SN_ℓ. This produces non-terminal paths such that both endpoints belong to $G[t_r]$;
4. connect-bridge the rest of the semi-terminal paths of S_r (at most $2(|F(T_\ell)| + |T_\ell|)$) using vertices of T_ℓ;
5. insert non-terminal paths obtained in Step 3 into the terminal paths of T_ℓ;

After computing the number of non-terminal, semi-terminal and terminal paths produced by procedure kPC_2_b, we conclude that:

$$\begin{aligned}|N_t| &= \max\{\mu,\ 0\} \\ |S_t| &= \min\{\sigma_r,\ \max\{\sigma_r - 2(|F(T_\ell)| + |T_\ell|),\ \delta(\sigma_r)\}\} \\ |T_t| &= |S_\ell| + \min\{\lfloor\tfrac{|S_r|-|S_\ell|}{2}\rfloor,\ |F(S_\ell \cup N_\ell)|\} + |T_\ell| + |T_r| + \tfrac{\sigma_r - |S_t|}{2}\end{aligned} \tag{5.5}$$

where

$$\begin{aligned}\sigma_r &= |S_r| - |S_\ell| - 2\min\{\lfloor\frac{|S_r| - |S_\ell|}{2}\rfloor,\ |F(S_\ell \cup N_\ell)|\} \\ \mu &= |N_r| - \pi_\ell - \min\{\max\{\min\{|F(S_\ell \cup N_\ell)| - \lfloor\frac{|S_r| - |S_\ell|}{2}\rfloor,\ 1\},\ 0\}, \\ &\quad \max\{\min\{|N_r| - \pi_\ell,\ \delta(|S_r| - |S_\ell|)\},\ 0\}\} - \frac{1}{2}\max\{2(|F(T_\ell)| + |T_\ell|) - \sigma_r,\ 0\}\end{aligned}$$

and

$$\pi_\ell = \max\{|F(S_\ell \cup N_\ell)| - \lceil\frac{|S_r| - |S_\ell|}{2}\rceil,\ 0\}.$$

In Eq. (5.5), σ_r is the number of semi-terminal paths of S_r that are not connected or bridged at Steps 1 and 2. Moreover, π_ℓ is the number of free vertices that belong to the set $S_\ell \cup N_\ell$ and are not used to bridge semi-terminal paths of S_r (at Step 2). Again, $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise. Note that at most $|F(T_\ell)| + |T_\ell|$ non-terminal paths can be inserted into the terminal paths of T_ℓ or the terminal paths can connect-bridge at most $2(|F(T_\ell)| + |T_\ell|)$ semi-terminal paths.

5.4 Correctness and Time Complexity

Let G be a cograph, $T(G)$ be the modified co-tree of G, and let $\mathcal{T}$ be a terminal set of G. Since our algorithm computes a kPC $\mathcal{P}'_{\mathcal{T}}(t)$ of $G[t]$ of size $\lambda'_{\mathcal{T}}(t)$ for each internal node $t \in T(G)$, and thus for the root $t = t_{root}$ of the tree $T(G)$, we need to prove that the constructed kPC $\mathcal{P}'_{\mathcal{T}}(t)$ is minimum. Obviously, the size $\lambda_{\mathcal{T}}(t)$ of a minimum kPC of the graph $G[t]$ is less than or equal to the size $\lambda'_{\mathcal{T}}(t)$ of the kPC constructed by our algorithm. According to Proposition 5.1, if the size of the kPC constructed by our algorithm is $\lambda'_{\mathcal{T}}(t) = \lceil \frac{|\mathcal{T}_t|}{2} \rceil$, then it is a minimum kPC. After performing simple computations on Eqs. (5.3)–(5.5), we get four specific values for the size $\lambda'_{\mathcal{T}}(t)$ of the path cover constructed by our algorithm, that is, by the kPC procedures 1, 2_a, and 2_b. More precisely, our algorithm returns a kPC of size $\lambda'_{\mathcal{T}}(t)$ equal to either $\frac{|\mathcal{T}_t|}{2} + 1$, $\lceil \frac{|\mathcal{T}_t|}{2} \rceil$, $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$, or $\lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|$; see Table 5.4. Recall that $\mathcal{T}_t$ denotes the set of terminal vertices in $V_t = V(G[t])$, while $F(V_\ell)$ (resp. $F(V_r)$) denotes the set of free vertices in $V_\ell = V(G[t_\ell])$ (resp. $V_r = V(G[t_r])$); t_ℓ and t_r are the left child and right child, respectively, of node t.

Let t be an internal node of $T(G)$ and let N_ℓ, S_ℓ and T_ℓ (resp. N_r, S_r and T_r) be the sets of non-terminal, semi-terminal and terminal paths, respectively, in $G[t_\ell]$ (resp. $G[t_r]$). For the case where $|S_\ell| = |T_r| = |S_r| = 0$ and $|N_\ell| = |V_r|$ our algorithm (procedure kPC_1) returns a kPC of the graph $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$. We prove the following lemma:

Lemma 5.2. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. If $|S_\ell| = |T_r| = |S_r| = 0$ and $|N_\ell| = |V_r|$, then the procedure kPC_1 returns a minimum kPC of $G[t]$ of size $\lambda_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$.*

Proof. Since we can construct a kPC of size $\lambda'_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$, then the size $\lambda_{\mathcal{T}}(t)$ of a minimum kPC is at most $\frac{|\mathcal{T}_t|}{2} + 1$. We will show that we can not construct a minimum kPC of size less than $\frac{|\mathcal{T}_t|}{2} + 1$, that is, we will show that $\lambda_{\mathcal{T}}(t) \geq \frac{|\mathcal{T}_t|}{2} + 1 \Leftrightarrow \lambda_{\mathcal{T}}(t) > \frac{|\mathcal{T}_t|}{2}$. Thus, we only need to prove that $\lambda_{\mathcal{T}}(t) \neq \frac{|\mathcal{T}_t|}{2}$. Note that by the assumption we have $\frac{|\mathcal{T}_t|}{2} = |T_\ell|$. We assume that $\lambda_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2}$, and, thus, $\lambda_{\mathcal{T}}(t) = |T_\ell|$. There exists at least one non-terminal path in $G[t_\ell]$; for otherwise $|N_\ell| = 0$, and thus $V_r = \emptyset$, a contradiction. We ignore the terminal paths from the minimum kPC of $G[t_\ell]$ and apply the algorithm described in [61] to $G[t]$. The resulting minimum kPC contains only one (non-terminal) path which either has both endpoints in $G[t_\ell]$ or it has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. This non-terminal path can not be inserted into a terminal path of $G[t_\ell]$ because it does not have both endpoints in $G[t_r]$. Thus, $\lambda_{\mathcal{T}}(t) = |T_\ell| + 1$, a contradiction. ∎

The previous lemma shows that if the size of the kPC returned by our subroutine process(t) for the graph $G[t]$ is $\lambda'_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$ (procedure kPC_1), then it is a minimum kPC. Moreover, if the size of the kPC returned by the process(t) is $\lceil \frac{|\mathcal{T}_t|}{2} \rceil$ (all the procedures), then it is obviously a minimum kPC of $G[t]$. We will prove that the size $\lambda'_{\mathcal{T}}(t)$ of the kPC $\mathcal{P}'_{\mathcal{T}}(t)$ that our subroutine process(t) returns is minimum.

Lemma 5.3. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. If the subroutine process(t) returns a kPC of $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \lceil \frac{|\mathcal{T}_t|}{2} \rceil$, then $\lambda'_{\mathcal{T}}(t) \geq \max\{\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Proof. Since $\lambda'_{\mathcal{T}}(t) = \lceil \frac{|\mathcal{T}_t|}{2} \rceil$, we have $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t)$, that is, the kPC that the subroutine process(t) returns is minimum. Thus, the proof follows from Lemma 5.1. ∎

Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. Furthermore, we assume that the conditions $|S_\ell| = |T_r| = |S_r| = 0$ and $|N_\ell| = |V_r|$ do not hold together. We consider the case where the subroutine process(t) returns a kPC $\mathcal{P}'_{\mathcal{T}}(t)$ of the graph $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$ (cases 1 and 2.a). We prove the following lemma.

Procedures	Size of k-fixed-endpoint PC
Procedure kPC_1	$\frac{\lvert\mathcal{T}_t\rvert}{2}+1$
All the procedures	$\lceil\frac{\lvert\mathcal{T}_t\rvert}{2}\rceil$
Procedure kPC_1 and Procedure kPC_2_a	$\lambda_{\mathcal{T}}(t_\ell)-\lvert F(V_r)\rvert$
Procedure kPC_2_b	$\lambda_{\mathcal{T}}(t_r)-\lvert F(V_\ell)\rvert$

Table 5.1: The size of the kPC that our algorithm returns in each case.

Lemma 5.4. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. If the subroutine process(t) returns a kPC of $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$, then $\lambda'_{\mathcal{T}}(t) > \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil,\ \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Proof. We consider the cases 1 and 2.a. In these cases, the size $\lambda'_{\mathcal{T}}(t)$ of the constructed kPC is computed using Eqs. (5.3) and (5.4) and the fact that $\lambda'_{\mathcal{T}}(t) = |N_t| + |S_t| + |T_t|$. After performing simple computations, we conclude that in these cases the subroutine process(t) returns $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$ if the following condition holds:

$$|N_\ell| + \lceil\frac{|S_\ell| - |S_r|}{2}\rceil > |F(V_r)| + |T_r| + \frac{\delta(|S_\ell| - |S_r|)}{2}. \tag{5.6}$$

We will show that (i) $\lambda'_{\mathcal{T}}(t) > \lceil\frac{|\mathcal{T}_t|}{2}\rceil$ and, (ii) $\lambda'_{\mathcal{T}}(t) > \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|$. We first consider the case where $\delta(|S_\ell| - |S_r|) = 0$. In this case either the values of $|S_\ell|$ and $|S_r|$ are both odd numbers or they are both even numbers. In any case, $|S_\ell| + |S_r|$ is an even number and thus, $|\mathcal{T}_t|$ is also an even number. Thus, according to Proposition 5.1 and since $G[t]$ is a cograph, we have $|\mathcal{T}_t| = |S_\ell| + |S_r| + 2|T_\ell| + 2|T_r|$.

(i) We first show that $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lceil\frac{|\mathcal{T}_t|}{2}\rceil$. From Eq. (5.6) and, since $2|T_r| + |S_r| = |\mathcal{T}_r|$ (see Proposition 5.1), we obtain

$$2|F(V_r)| + |\mathcal{T}_r| < 2|N_\ell| + |S_\ell|. \tag{5.7}$$

By Proposition 5.1 and Eq. (5.7) we have

$$2(|N_\ell| + |S_\ell| + |T_\ell| - |F(V_r)|) > 2|F(V_r)| + |\mathcal{T}_r| + |S_\ell| + 2|T_\ell| - 2|F(V_r)| = |\mathcal{T}_\ell| + |\mathcal{T}_r| = |\mathcal{T}_t|.$$

Since $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| = |N_\ell| + |S_\ell| + |T_\ell| - |F(V_r)|$, it follows that $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lceil\frac{|\mathcal{T}_t|}{2}\rceil$.

(ii) We next show that $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|$. According to Proposition 5.2, we have $\lambda_{\mathcal{T}}(t_r) - |F(V_\ell)| \leq |F(V_r)| + |\mathcal{T}_r| - |F(V_\ell)|$. From Eq. (5.7) and since

$$|N_\ell| \leq |F(N_\ell)| \Leftrightarrow |N_\ell| \leq |F(V_\ell)| \Leftrightarrow |N_\ell| - |F(V_\ell)| \leq 0,$$

we obtain $|F(V_r)| + |\mathcal{T}_r| - |F(V_\ell)| < 2|N_\ell| + |S_\ell| - |F(V_r)| - |F(V_\ell)| \leq 2|N_\ell| + |S_\ell| + |T_\ell| - |F(V_r)| - |F(V_\ell)| = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| + |N_\ell| - |F(V_\ell)| \leq \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$.

Similarly, we can prove that $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lceil\frac{|\mathcal{T}_t|}{2}\rceil$ and $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|$, for the case where $\delta(|S_\ell| - |S_r|) = 1$, and, thus, $\lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil,\ \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\}$. ∎

Using arguments similar to those used to prove Lemma 5.4, we can show that if the subroutine process(t) returns a kPC of $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|$ (case 2.b), then $\lambda'_{\mathcal{T}}(t) > \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil,\ \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|\}$; note that, in this case we have $|N_r| + \lceil\frac{|S_r|-|S_\ell|}{2}\rceil > |F(V_\ell)| + |T_\ell| + \frac{\delta(|S_r|-|S_\ell|)}{2}$. Thus, we have proved the following result.

Lemma 5.5. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. The subroutine process(t) returns a kPC $\mathcal{P}_{\mathcal{T}}(t)$ of $G[t]$ of size*

$$\lambda'_{\mathcal{T}}(t) = \begin{cases} \frac{|\mathcal{T}_t|}{2} + 1 & \text{if } |S_\ell| = |\mathcal{T}_r| = 0 \text{ and } |N_\ell| = |V_r|, \\ \\ \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil, \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\} & \text{otherwise.} \end{cases}$$

Obviously, a minimum kPC of the graph $G[t]$ is of size $\lambda_{\mathcal{T}}(t) \leq \lambda'_{\mathcal{T}}(t)$. On the other hand, we have proved a lower bound for the size $\lambda_{\mathcal{T}}(t)$ of a minimum kPC of the graph $G[t]$ (see Lemma 5.1), namely, $\lambda_{\mathcal{T}}(t) \geq \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil,\ \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\}$. It follows that $\lambda'_{\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t)$, and, thus, we can state the following result.

Lemma 5.6. *Subroutine process(t) returns a minimum kPC $\mathcal{P}_{\mathcal{T}}(t)$ of the graph $G[t]$, for every internal S-node $t \in T(G)$.*

Since the above result holds for every S-node t of the modified co-tree $T(G)$, it also holds when t is the root of $T(G)$ and $\mathcal{T}_t = \mathcal{T}$. Thus, the following theorem holds:

Theorem 5.1. *Let G be a cograph and let $\mathcal{T}$ be a terminal set of G. Let t be the root of the modified co-tree $T(G)$, and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kPC of $G[t_\ell]$ and $G[t_r]$, respectively. Algorithm Minimum_kPC correctly computes a minimum kPC of $G = G[t]$ with respect to $\mathcal{T} = \mathcal{T}_t$ of size $\lambda_{\mathcal{T}} = \lambda_{\mathcal{T}}(t)$, where*

$$\lambda_{\mathcal{T}}(t) = \begin{cases} \lambda_{\mathcal{T}}(t_r) + \lambda_{\mathcal{T}}(t_\ell) & \text{if } t \text{ is a P-node,} \\ \frac{|\mathcal{T}_t|}{2} + 1 & \text{if } t \text{ is an S-node and} \\ & |S_\ell| = |\mathcal{T}_r| = 0 \text{ and } |N_\ell| = |V_r|, \\ \max\{\lceil\frac{|\mathcal{T}_t|}{2}\rceil, \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|, \lambda_{\mathcal{T}}(t_r) - |F(V_\ell)|\} & \text{otherwise.} \end{cases}$$

Let G be a cograph on n vertices and m edges, $\mathcal{T}$ be a terminal set, and let t be an S-node of the modified co-tree $T(G)$. From the description of the algorithm we can easily conclude that a minimum kPC $\mathcal{P}_{\mathcal{T}}(t)$ of $G[t]$ can be constructed in $O(E(G[t]))$ time, since we use at most $|V(G[t_\ell])| \cdot |V(G[t_r])|$ edges to connect the paths of the minimum kPCs of the graphs $G[t_\ell]$ and $G[t_r]$; in the case where t is a P-node a minimum kPC is constructed in $O(1)$ time. Thus, the time needed by the subroutine process(t) to compute a minimum kPC in the case where t is the root of the tree $T(G)$ is $O(n+m)$; moreover, through the execution of the subroutine no additional space is needed. The construction of the co-tree $T_{co}(G)$ of G needs $O(n+m)$ time and it requires $O(n)$ space [22, 24]. Furthermore, the binarization process of the co-tree, that is, the construction of the modified co-tree $T(G)$, takes $O(n)$ time. Hence, we can state the following result.

Theorem 5.2. *Let G be a cograph on n vertices and m edges and let $\mathcal{T}$ be a terminal set of G. A minimum k-fixed-endpoint path cover $\mathcal{P}_{\mathcal{T}}$ of G can be computed in $O(n+m)$ time and space.*

5.5 Concluding Remarks

This chapter presents a simple linear-time algorithm for the k-fixed-endpoint path cover problem on cographs. Given a cograph G and a set of vertices $\mathcal{T}$, our algorithm constructs a minimum k-fixed-endpoint path cover of G that contains a large number of terminal paths.

Thus, it is worth investigating the existence of a linear-time algorithm for finding a minimum k-fixed-endpoint path cover on cographs such that it contains a large number of semi-terminal paths; we pose it as an open problem.

It would also be interesting to define a variant of the k-fixed-endpoint path cover problem, which would include a pair of terminal sets $\mathcal{T}_1$ and $\mathcal{T}_2$. In this variant, we want to compute a minimum path cover such that all the terminal paths have one endpoint in $\mathcal{T}_1$ and the other in $\mathcal{T}_2$. Then, we could use this extended k-fixed-endpoint path cover problem to solve problem B (see Section 5.1). This problem is defined and solved for the class of cographs in Chapter 7.

Finally, an interesting open question would also be to see if the k-fixed-endpoint path cover problem can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. This promises to be an interesting area for further research.

Chapter 6

The k-fixed-endpoint-set Path Cover Problem and its Polynomial Solution on Cographs

6.1 Introduction

In this chapter, we study a generalization of the path cover problem, namely, the k-fixed-endpoint-set path cover problem, or kFSPC for short. Given a graph G and k disjoint subsets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of $V(G)$, a k-fixed-endpoint-set path cover of G is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are all endpoints of the paths in $\mathcal{P}$ and two vertices $u, v \in \mathcal{T}$ belong to the same path of $\mathcal{P}$ if both u, v belong to the same set $\mathcal{T}^i$, $i \in [1, k]$. Note that, if $\mathcal{T}^i$, $\forall i \in [1, k]$ is empty, the stated problem coincides with the classical path cover problem, while the 1-fixed-endpoint-set path cover problem coincides with the k-fixed-endpoint path cover problem (kPC). Thus, the kFSPC problem generalizes the kPC problem and also the 1HP and 2HP problems, which have been proved to be NP-complete even for small classes of graphs. We show that the kFSPC problem can be solved in linear time on the class of cographs. The proposed linear-time algorithm is simple and requires linear space. We next define the kFSPC problem.

Problem kFSPC. Let G be a graph and let $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be disjoint sets of vertices of $V(G)$. A *k-fixed-endpoint-set path cover* $\mathcal{P}$ of the graph G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a path cover of G such that all vertices in $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are endpoints of paths in $\mathcal{P}$ and two vertices $u, v \in \mathcal{T}$

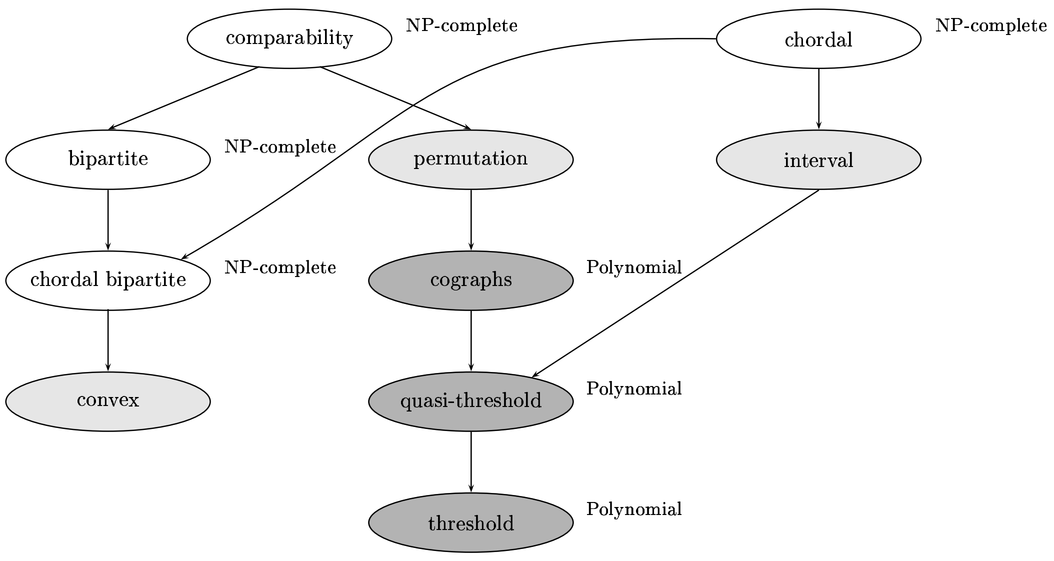


Figure 6.1: The complexity status (NP-complete, unknown, polynomial) of the kFSPC problem for some graph subclasses of comparability and chordal graphs. $A \to B$ indicates that class A contains class B.

belong to the same path of $\mathcal{P}$ if both u, v belong to the same set $\mathcal{T}^i$, $i \in [1,k]$; a ***minimum k-fixed-endpoint-set path cover*** of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a k-fixed-endpoint-set path cover of G with minimum cardinality; the ***k-fixed-endpoint-set path cover problem*** (kFSPC) is to find a minimum k-fixed-endpoint-set path cover of the graph G.

Contribution. In this chapter, we show that the k-fixed-set path cover problem (kFSPC) has a polynomial-time solution in the class of complement reducible graphs, or cographs [24]. More precisely, we establish a lower bound on the size of a minimum k-fixed-set path cover of a cograph G on n vertices and m edges. We then define path operations, and prove structural properties for the paths of such a path cover, which enable us to describe a simple algorithm for the kFSPC problem. The proposed algorithm runs in time linear in the size of the input graph G, that is, in $O(n+m)$ time, and requires linear space. Figure 6.1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status of the kFSPC problem on these classes; for definitions of the classes shown, see [13, 40].

The proposed algorithm for the kFSPC problem can also be used to solve the 1HP and 2HP problems on cographs within the same time and space complexity. Moreover, we have designed our algorithm so that it produces a minimum k-fixed-set path cover of a cograph G that contains a large number of paths with both endpoints in $\mathcal{T}^i$, $i \in [1,k]$ (we can easily find a graph G so that G admits two minimum k-fixed-set path covers with different numbers of paths having both endpoints in $\mathcal{T}^i$, $i \in [1,k]$).

6.2 Theoretical Framework

The cographs admit a tree representation unique up to isomorphism. Specifically, we can associate with every cograph G a unique rooted tree $T_{co}(G)$ called the co-tree (or, modular decomposition tree [68]), which we can construct sequentially in linear time [22, 24]. The co-tree forms the basis for fast algorithms for problems such as isomorphism, coloring, clique detection, clusters, minimum weight dominating sets [13, 22], and also for the path cover problem [61, 70].

For convenience and ease of presentation, we binarize the co-tree $T_{co}(G)$ in such a way that each of its internal nodes has exactly two children [61, 70]. We shall refer to the binarized version of $T_{co}(G)$ as the modified co-tree of G and will denote it by $T(G)$. Thus, the left and right child of an internal node t of $T(G)$ will be denoted by t_ℓ and t_r, respectively. Let t be an internal node of $T(G)$. Then $G[t]$ is the subgraph of G induced by the subset V_t of the vertex set $V(G)$, which contains all the vertices of G that have as common ancestor in $T(G)$ the node t. For simplicity, we will denote by V_ℓ and V_r the vertex sets $V(G[t_\ell])$ and $V(G[t_r])$, respectively.

Let G be a cograph, $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be k disjoint sets of vertices of $V(G)$ and let $\mathcal{P}_{k\mathcal{T}}(G)$ be a minimum k-fixed-set path cover of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of size $\lambda_{k\mathcal{T}}$; note that the size of $\mathcal{P}_{k\mathcal{T}}(G)$ is the number of paths it contains. The vertices of the sets $\mathcal{T}^i$, $1 \leq i \leq k$ are called *terminal* vertices , and the sets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ are called the *terminal sets* of G, while those of $V(G) - (\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k)$ are called *non-terminal or free* vertices. Furthermore, we say that a vertex u is a terminal vertex and denote it by $u \in \mathcal{T}$ if it belongs to any $\mathcal{T}^i$, $1 \leq i \leq k$. Thus, the set $\mathcal{P}_{k\mathcal{T}}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices belonging to the same set, that is, $u \in \mathcal{T}^i$ and $v \in \mathcal{T}^i$, $i \in [1, k]$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}^i$, $1 \leq i \leq k$ and the other in $V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}^i$, $1 \leq i \leq k$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$; if P_f consists of only one vertex, say, u, then $u \in V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$.

The set of the non-terminal paths in a minimum kFSPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Furthermore, we denote by S_i (resp. T_i) a set of semi-terminal (resp. terminal) paths having a terminal endpoint (resp. both endpoints) that belongs to the terminal set $\mathcal{T}^i$. The following equation holds.

$$\lambda_{k\mathcal{T}} = |N| + |S| + |T| \tag{6.1}$$

From the definition of the k-fixed-set path cover problem (kFSPC), we can easily conclude that the number of paths in a minimum kFSPC cannot be less than the number of the terminal vertices divided by two. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T|$. Thus, we have the following proposition, which also holds for general graphs:

Proposition 6.1. *Let G be a cograph and let $\mathcal{T}^1, \ldots, \mathcal{T}^k$ be disjoint subsets of $V(G)$. Then $\lambda_{k\mathcal{T}} \geq \Sigma_{i=1}^{k} \lceil \frac{|\mathcal{T}^i|}{2} \rceil$.*

Clearly, the size of a kFSPC of a cograph G, as well as the size of a minimum kFSPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_{k\mathcal{T}} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G; hereafter, $F(V) = F(V(G))$. Furthermore, let $\mathcal{P}$ be a set of paths and let $V_{\mathcal{P}}$ denote the set of vertices belonging to the paths of the set $\mathcal{P}$; hereafter, $F(\mathcal{P}) = F(V_{\mathcal{P}})$. Then, if $\mathcal{T}^1, \ldots, \mathcal{T}^k$ are disjoint subsets of $V(G)$, we have $\lambda_{k\mathcal{T}} \leq |F(V)| + |\mathcal{T}^1| + \ldots + |\mathcal{T}^k|$.

Let t be an internal node of the tree $T(G)$, that is, t is either an S-node or a P-node [68]. Then $\lambda_{k\mathcal{T}}(t)$ denotes the number of paths in a minimum kFSPC of the graph $G[t]$ with respect to $\mathcal{T}_t^1, \ldots, \mathcal{T}_t^k$, where $\mathcal{T}_t^i$, $1 \leq i \leq k$ are the terminal vertices of $\mathcal{T}^i$ of the graph $G[t]$. Let t_ℓ and t_r be the left and the right child of node t, respectively. We denote by $\mathcal{T}_\ell^i$ and $\mathcal{T}_r^i$ the terminal vertices of $\mathcal{T}^i$ in V_ℓ and V_r, respectively, where $V_\ell = V(G[t_\ell])$ and $V_r = V(G[t_r])$. Furthermore, we denote by $\mathcal{T}_\ell$ and $\mathcal{T}_r$ the set containing the terminal vertices belonging in any $\mathcal{T}^i$ in V_ℓ and V_r, respectively. Let N_ℓ, S_ℓ and T_ℓ be the sets of the non-terminal, semi-terminal and terminal paths in a minimum kFSPC of $G[t_\ell]$, respectively. Similarly, let N_r, S_r and T_r be the sets of the non-terminal, semi-terminal and terminal paths in a minimum kFSPC of $G[t_r]$, respectively. Clearly, S_ℓ^i and T_ℓ^i (resp. S_r^i and T_r^i) denote the sets of the semi-terminal and terminal paths in a minimum kFSPC of $G[t_\ell]$ (resp. $G[t_r]$) with terminal vertices belonging to $\mathcal{T}^i$, respectively. Obviously, Eq. (6.1) holds for $G[t]$ as well, with t being either an S-node or a P-node, that is,

$$\lambda_{k\mathcal{T}}(t) = |N_t| + |S_t| + |T_t| \tag{6.2}$$

where N_t, S_t and T_t are the sets of the non-terminal, the semi-terminal and the terminal paths, respectively, in a minimum kFSPC of $G[t]$, that is in $\mathcal{P}_{k\mathcal{T}}(t)$. If t is a P-node, then $\mathcal{P}_{k\mathcal{T}}(t) = \mathcal{P}_{k\mathcal{T}}(t_\ell) \cup \mathcal{P}_{k\mathcal{T}}(t_r)$, where $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ are minimum kFSPCs corresponding to $G[t_\ell]$ and $G[t_r]$, respectively, and $\lambda_{k\mathcal{T}}(t) = \lambda_{k\mathcal{T}}(t_\ell) + \lambda_{k\mathcal{T}}(t_r)$. Furthermore, in the case where t is a P-node, we have

$$\begin{aligned} |N_t| &= |N_\ell| + |N_r| \\ |S_t| &= |S_\ell| + |S_r| \\ |T_t| &= |T_\ell| + |T_r| \end{aligned}$$

Thus, we focus on computing a minimum kFSPC of the graph $G[t]$ for the case where t is an S-node. We next define four operations on paths of a minimum kFSPC of the graphs $G[t_\ell]$ and $G[t_r]$, namely *break*, *connect*, *bridge* and *insert* operations; these operations are illustrated in Fig. 6.2.

- *Break* operation: Let $P = [p_1, p_2, \ldots, p_s]$ be a path of $\mathcal{P}_{k\mathcal{T}}(t_r)$ or $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ of length s. We say that we *break* the path P in two paths, say, P_1 and P_2, if we delete an arbitrary edge of P, say the edge $p_i p_{i+1}$ $(1 \leq i < s)$, in order to obtain two paths which are $P_1 = [p_1, \ldots, p_i]$ and $P_2 = [p_{i+1}, \ldots, p_s]$. Note that we can break the path P in at most s trivial paths.

- *Connect* operation: Let P_1 be a non-terminal or a semi-terminal path of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_r)$) and let P_2 be a non-terminal or a semi-terminal path of $\mathcal{P}_{k\mathcal{T}}(t_r)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_\ell)$). We say that we *connect* the path P_1 with the path P_2, if we add an edge which joins two free endpoints of the two paths. Note that if $P_1 \in S_\ell$ (resp. $P_1 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^i$ then, if P_2 is also a semi-terminal path, $P_2 \in S_r$ (resp. $P_2 \in S_\ell$) with a terminal endpoint belonging to $\mathcal{T}^j$, $i \neq j$.

- *Bridge* operation: Let P_1 and P_2 be two paths of the set $N_\ell \cup S_\ell^1 \cup S_\ell^2$ (resp. $N_r \cup S_r^1 \cup S_r^2$) and let P_3 be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *bridge* the two paths P_1 and P_2 using path P_3 if we connect a free endpoint of P_1 with one endpoint of P_3 and a free endpoint of P_2 with the other endpoint of P_3. The result is a path having both endpoints in $G[t_\ell]$ (resp. $G[t_r]$). Note that if $P_1 \in S_\ell$ (resp. $P_1 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^i$ then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell$ (resp. $P_2 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^j$, $i \neq j$.

- *Insert* operation: Let $P_1 = [t_1, p_1, \ldots, p_1', t_1']$ be a path in $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_r)$) and let $P_2 = [p_2, \ldots, p_2']$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *insert* the path P_2 into P_1, if we replace the first edge of P_1, that is, the edge $t_1 p_1$, with the path $[t_1, p_2, \ldots, p_2', p_1]$. Thus, the resulting path is $P_1 = [t_1, p_2, \ldots, p_2', p_1, \ldots, p_1', t_1']$. Note that we can replace every edge of P_1 so that we can insert at most $|F'(\{P_1\})| + 1$ non-terminal paths, where $F'(\{P_1\})$ is the set of the free internal vertices belonging to the path P_1. If the path $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_1^r, \ldots, p_1', t_1']$ is constructed by connecting a path of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$, say, $P_\ell = [t_1, p_1, \ldots, p_1^\ell]$ with a path of $\mathcal{P}_{k\mathcal{T}}(t_r)$, say, $P_r = [p_1^r, \ldots, p_1', t_1']$, then it obviously has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. In this case,

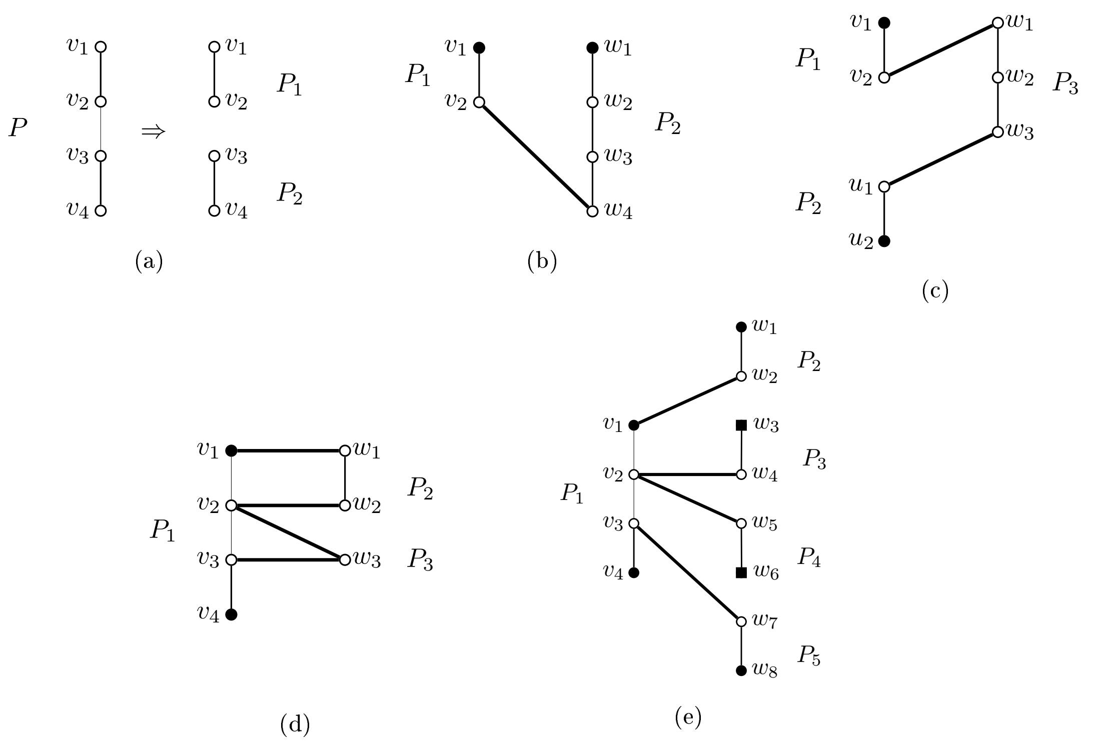


Figure 6.2: Illustrating (a) break, (b) connect, (c) bridge, (d) insert, and (e) connect-bridge operations; the vertices denoted by black-circles belong to $\mathcal{T}^i$, while the vertices denoted by black-squares belong to $\mathcal{T}^j$, $i \neq j$.

if $P_2 \in N_\ell$ (resp. N_r) we can only replace the edges of P_1 that belong to $G[t_r]$ (resp. $G[t_\ell]$). On the other hand, if P_2 has one endpoint, say, p_2, in N_ℓ and the other, say, p_2', in N_r, we insert P_2 into P_1 as follows: $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_2', \ldots, p_2, p_1^r, \ldots, p_1', t_1']$.

We can also combine the operations connect and bridge to perform a new operation which we call a *connect-bridge* operation; such an operation is depicted in Fig. 6.2(e) and is defined below.

- *Connect-Bridge* operation: Let $P_1 = [t_1, p_1, \ldots, p_z, t_1']$ be a terminal path of the set T_ℓ (resp. T_r), where $t_1 \in \mathcal{T}^i$ and $t_1' \in \mathcal{T}^j$, $i \neq j$, and let $P_2, P_3, \ldots, P_{\frac{s+1}{2}}$ be semi-terminal paths of the set S_r (resp. S_ℓ) with terminal endpoints belonging to pairwise disjoint sets $\mathcal{T}^p$ and $P_{\frac{s+1}{2}+1}, \ldots, P_s$ be semi-terminal paths of the set S_r (resp. S_ℓ) with terminal endpoints belonging to pairwise disjoint sets $\mathcal{T}^{p'}$, where s is odd and $3 \leq s \leq 2z+3$. Let P_2 have a terminal endpoint belonging to $\mathcal{T}^{i'}$, $i' \neq i$, and let P_s have a terminal endpoint belonging to $\mathcal{T}^{j'}$, $j' \neq j$. We say that we *connect-bridge* the paths $P_2, P_3, \ldots, P_s$ using vertices of P_1, if we perform the following operations: (i) connect the path P_2 with the path $[t_1]$; (ii) bridge $r = \frac{s-3}{2}$ pairs of different semi-terminal paths using vertices $p_1, p_2, \ldots, p_r$; and (iii) connect the path $[p_{r+1}, \ldots, p_z, t_1']$ with the last semi-terminal path P_s.

The Connect-Bridge operation produces two paths having one endpoint in $G[t_\ell]$ and the other endpoint in $G[t_r]$ and $\frac{s-3}{2}$ paths having both endpoints in $G[t_r]$ (resp. $G[t_\ell]$).

6.3 The Algorithm

We next present an optimal algorithm for the kFSPC problem on cographs. Our algorithm takes as input a cograph G and k disjoint subsets $\mathcal{T}^1, \ldots, \mathcal{T}^k$ of its vertices, where $|\mathcal{T}^i| \leq 2$, $\forall i \in [1, k]$, and finds the paths of a minimum kFSPC of G in linear time; it works as follows:

Algorithm Minimum_kFSPC

Input: a cograph G and k disjoint subsets $\mathcal{T}^1, \ldots, \mathcal{T}^k$ of $V(G)$;

Output: a minimum kFSPC $\mathcal{P}_{k\mathcal{T}}(G)$ of the cograph G;

---

1. Construct the co-tree $T_{co}(G)$ of G and make it binary; let $T(G)$ be the resulting tree;
2. Execute the subroutine *process*(*root*), where *root* is the root node of the tree $T(G)$; the minimum kFSPC $\mathcal{P}_{k\mathcal{T}}(root) = \mathcal{P}_{k\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

Algorithm 4: Algorithm Minimum_kFSPC

where the description of the subroutine *process*( ) is as follows:

process (node t)

Input: node t of the modified co-tree $T(G)$ of the input graph G.

Output: a minimum kFSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of the cograph $G[t]$.

```
1. if $t$ is a leaf
   then return($\{u\}$), where $u$ is the vertex associated with the leaf $t$;
   else {$t$ is an internal node that has a left and a right child denoted by $t_\ell$ and $t_r$, resp.}
        $\mathcal{P}_{k\mathcal{T}}(t_\ell) \leftarrow process(t_\ell)$;
        $\mathcal{P}_{k\mathcal{T}}(t_r) \leftarrow process(t_r)$;

2. if $t$ is a P-node
   then return($\mathcal{P}_{k\mathcal{T}}(t_\ell) \cup \mathcal{P}_{k\mathcal{T}}(t_r)$);

3. if $t$ is an S-node
   then call procedure $kFSPC$;
```

We next describe procedure $kFSPC$. Let b_r (resp. b_ℓ) be the number of vertices needed to bridge the semi-terminal paths of S_ℓ (resp. S_r) that cannot be connected to semi-terminal paths of S_r. Thus, $b_r = \sum_{i=1}^{k} \lfloor \frac{\max\{S_\ell^i - S_r^i, 0\}}{2} \rfloor$ and $b_\ell = \sum_{i=1}^{k} \lfloor \frac{\max\{S_r^i - S_\ell^i, 0\}}{2} \rfloor$. Let SN_r (resp. SN_ℓ) be the set of non-terminal paths obtained by breaking paths of the set $S_r \cup N_r$ (resp. $S_r \cup N_r$). We can construct the paths of a kFSPC using the following procedure:

Procedure kFSPC

1. if $|N_r| - b_r > |N_\ell| - b_\ell$ then
   - 1.1 break the paths of $N_\ell \cup S_\ell$ in order to obtain a set SN_ℓ of $\min\{b_\ell + |N_r| - b_r, F(S_\ell \cup N_\ell)\}$ paths;
   - 1.2 break the paths of $N_r \cup S_r$ in order to obtain a set SN_r of $\min\{b_r, F(S_r \cup N_r)\}$ paths;

```
    1.3 connect paths of $S_\ell$ with paths of $S_r$;
    1.4 bridge semi-terminal paths of $S_\ell$ (resp. $S_r$) using paths of $SN_r$ (resp. $SN_\ell$);
    1.5 bridge non-terminal paths of $V_r$ using non-terminal paths of $V_\ell$ and, if we obtain one path connect the last non-terminal path of $V_\ell$ to a non-terminal path of $V_r$. The resulting path $p$ has one endpoint in $V_\ell$ and the other in $V_r$;
    1.6 if $\exists$ a non-terminal path $p$ with one endpoint in $V_r$ and the other in $V_\ell$ and if there exist paths created at Step 1.3, insert $p$ into one of them; else insert $p$ into a path created at Step 1.4;
2. if $|N_r| - b_r < |N_\ell| - b_\ell$ then
    2.1 break the paths of $N_r \cup S_r$ in order to obtain a set $SN_r$ of $\min\{b_r + |N_\ell| - b_\ell, F(S_r \cup N_r)\}$ paths;
    2.2 break the paths of $N_\ell \cup S_\ell$ in order to obtain a set $SN_\ell$ of $\min\{b_\ell, F(S_\ell \cup N_\ell)\}$ paths;
    2.3 connect paths of $S_\ell$ with paths of $S_r$;
    2.4 bridge semi-terminal paths of $S_\ell$ (resp. $S_r$) using paths of $SN_r$ (resp. $SN_\ell$);
    2.5 bridge non-terminal paths of $V_\ell$ using non-terminal paths of $V_r$ and, if we obtain one path connect the last non-terminal path of $V_r$ to a non-terminal path of $V_\ell$. The resulting path $p$ has one endpoint in $V_\ell$ and the other in $V_r$;
    2.6 if $\exists$ a non-terminal path $p$ with one endpoint in $V_r$ and the other in $V_\ell$ and if there exist paths created at Step 2.3, insert $p$ into one of them; else insert $p$ into a path created at Step 2.4;
3. if $|N_r| - b_r = |N_\ell| - b_\ell$ then
    3.1 break the paths of $N_r \cup S_r$ in order to obtain a set $SN_r$ of $\min\{b_r, F(S_r \cup N_r)\}$ paths;
    3.2 break the paths of $N_\ell \cup S_\ell$ in order to obtain a set $SN_\ell$ of $\min\{b_\ell, F(S_\ell \cup N_\ell)\}$ paths;
    3.3 connect paths of $S_\ell$ with paths of $S_r$;
    3.4 bridge semi-terminal paths of $S_\ell$ (resp. $S_r$) using paths of $SN_r$ (resp. $SN_\ell$);
    3.5 bridge non-terminal paths of $V_r$ using non-terminal paths of $V_\ell$ and connect the last non-terminal path of $V_\ell$ to a non-terminal path of $V_r$. The constructed path $p$ has one endpoint in $V_\ell$ and the other in $V_r$.
    3.6 if there exist paths created at Step 3.3, insert $p$ into one of them; else insert $p$ into a path created at Step 3.4;
4. $\forall i \in [1, k]$ find the longest path $p_1 = [t_1, v_1, \ldots, v'_1, t'_1]$ of $T^i_\ell$ and the longest path $p_2 = [t_2, v_2, \ldots, v'_2, t'_2]$ of $T^i_r$:
    4.1 break $p_1$ and $p_2$ to obtain the paths $p_1 = [t_1]$, $p'_1 = [v_1, \ldots, v'_1, t'_1]$, $p_2 = [t_2]$ and $p'_2 = [v_2, \ldots, v'_2, t'_2]$;
    4.2 connect $p_1$ to $p_2$;
    4.3 break $p'_1$ and $p'_2$ in order to obtain free vertices to bridge semi-terminal paths of $S_\ell$ or $S_r$. Let $p''_1$ and $p''_2$ be the resulting paths;
    4.4 connect $p''_1$ and $p''_2$;
5. connect-bridge semi-terminal paths of $S_\ell$ (resp. $S_r$) with paths of $T_r$ (resp. $T_\ell$);
6. use the free vertices of the paths of $T^i_r$ (resp. $T^i_\ell$) to bridge semi-terminal paths of $S^j_\ell$ (resp. $S^j_r$), $i \neq j$. Let $\Upsilon^i_r$ (resp. $\Upsilon^i_\ell$) be the number of used terminal paths of $T^i_r$ (resp. $T^i_\ell$);
7. insert non-terminal paths of $V_\ell$ (resp. $V_r$) into paths of $T_r$ (resp. $T_\ell$);
```

6.4 Correctness and Time Complexity

Let G be a cograph, $T(G)$ be the modified co-tree of G, and let $\mathcal{T}^i$ $i \in [1,k]$ be the terminal sets of G. Since our algorithm computes a kFSPC $\mathcal{P}'_{k\mathcal{T}}(t)$ of $G[t]$ of size $\lambda'_{k\mathcal{T}}(t)$ for each internal node $t \in T(G)$, and thus for the root $t = t_{root}$ of the tree $T(G)$, we need to prove that the constructed kFSPC $\mathcal{P}'_{k\mathcal{T}}(t)$ is minimum. Obviously, the size $\lambda_{k\mathcal{T}}(t)$ of a minimum kFSPC of the graph $G[t]$ is less than or equal to the size $\lambda'_{k\mathcal{T}}(t)$ of the kFSPC constructed by our algorithm. After performing simple computations we get four specific values for the size $\lambda'_{k\mathcal{T}}(t)$ of the kFSPC constructed by our algorithm, that is, by the procedure kFSPC. More precisely, if t is an internal S-node of $T(G)$, our algorithm returns a kFSPC of size $\lambda'_{k\mathcal{T}}(t)$ equal to $\max\{\lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|,\ \Sigma_{i=1}^{k}\lceil\frac{|\mathcal{T}^i|}{2}\rceil,\ \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k}\Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu}\Upsilon_\ell^i\}$, where $\mathcal{T}^i$, $1 \le i \le \nu$ (resp. $\nu+1 \le i \le k$) denotes the terminal set $\mathcal{T}^i$ for which $|\mathcal{T}_\ell^i| \ge |\mathcal{T}_r^i|$ (resp. $|\mathcal{T}_\ell^i| < |\mathcal{T}_r^i|$) and $\sum_{i=\nu+1}^{k}\Upsilon_r^i$ (resp. $\sum_{i=1}^{\nu}\Upsilon_\ell^i$) denotes the number of paths of T_r^i (resp. T_ℓ^i) that Algorithm Minimum_kFSPC breaks into semi-terminal paths. For the case where $|S_\ell| = |T_r| = |S_r| = 0$ and $|N_\ell| = |V_r|$ our algorithm returns a kFSPC of the graph $G[t]$ of size $\lambda'_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$. We prove the following lemma:

Lemma 6.1. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{\mathcal{T}}(t_r)$ be a minimum kFSPC of $G[t_\ell]$ and $G[t_r]$, respectively. If $|S_\ell| = |\mathcal{T}_r| = 0$ and $|N_\ell| = |V_r|$, then Algorithm Minimum_kFSPC returns a minimum kFSPC of $G[t]$ of size $\lambda_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$.*

Proof. Since we can construct a kFSPC of size $\lambda'_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2} + 1$, then the size $\lambda_{\mathcal{T}}(t)$ of a minimum kFSPC is at most $\frac{|\mathcal{T}_t|}{2} + 1$. We will show that we can not construct a minimum kFSPC of size less than $\frac{|\mathcal{T}_t|}{2} + 1$, that is, we will show that $\lambda_{\mathcal{T}}(t) \ge \frac{|\mathcal{T}_t|}{2} + 1 \Leftrightarrow \lambda_{\mathcal{T}}(t) > \frac{|\mathcal{T}_t|}{2}$. Thus, we only need to prove that $\lambda_{\mathcal{T}}(t) \ne \frac{|\mathcal{T}_t|}{2}$. Note that by the assumption we have $\frac{|\mathcal{T}_t|}{2} = |T_\ell|$. We assume that $\lambda_{\mathcal{T}}(t) = \frac{|\mathcal{T}_t|}{2}$, and, thus, $\lambda_{\mathcal{T}}(t) = |T_\ell|$. There exists at least one non-terminal path in $G[t_\ell]$; for otherwise $|N_\ell| = 0$, and thus $V_r = \emptyset$, a contradiction. We ignore the terminal paths from the minimum kFSPC of $G[t_\ell]$ and apply the algorithm described in [61] to $G[t]$. The resulting minimum kFSPC contains only one (non-terminal) path which either has both endpoints in $G[t_\ell]$ or it has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. This non-terminal path can not be inserted into a terminal path of $G[t_\ell]$ because it does not have both endpoints in $G[t_r]$. Thus, $\lambda_{\mathcal{T}}(t) = |T_\ell| + 1$, a contradiction. ∎

We next establish a lower bound on the size $\lambda_{\mathcal{T}}(t)$ of a minimum kFSPC $\mathcal{P}_{\mathcal{T}}(t)$ of a graph $G[t]$. More precisely, we prove the following lemma.

Lemma 6.2. *Let t be an internal node of $T(G)$ and let $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ be a minimum kFSPC of $G[t_\ell]$ and $G[t_r]$, respectively. Let $\lambda_\ell^i = |S_\ell^i| + |T_\ell^i|$ and $\lambda_r^i = |S_r^i| + |T_r^i|$. Then $\lambda_{k\mathcal{T}}(t) \ge \max\{\lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|,\ \Sigma_{i=1}^{k}\lceil\frac{|\mathcal{T}^i|}{2}\rceil,\ \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k}\Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu}\Upsilon_\ell^i\}$, where $\mathcal{T}^i$, $1 \le i \le \nu$ (resp. $\nu+1 \le i \le k$) denotes the terminal set $\mathcal{T}^i$ for which $|\mathcal{T}_\ell^i| \ge |\mathcal{T}_r^i|$ (resp. $|\mathcal{T}_\ell^i| < |\mathcal{T}_r^i|$) and $\sum_{i=\nu+1}^{k}\Upsilon_r^i$ (resp. $\sum_{i=1}^{\nu}\Upsilon_\ell^i$) denotes the number of paths of T_r^i (resp. T_ℓ^i) that Algorithm Minimum_kFSPC breaks into semi-terminal paths.*

Proof. Clearly, according to Proposition 6.1 and since $G[t]$ is a cograph, we have $\lambda_{\mathcal{T}}(t) \ge \Sigma_{i=1}^{k}\lceil\frac{|\mathcal{T}^i|}{2}\rceil$. We prove that $\lambda_{k\mathcal{T}}(t) \ge \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$. Assume that $\lambda_{k\mathcal{T}}(t) < \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$. Consider removing from this path cover all the vertices in V_r. What results is a set of paths which is clearly a kFSPC for $G[t_\ell]$. Since the removal of a free vertex in $F(V_r)$ will increase the number of paths by at most one, we obtain a kFSPC of $G[t_\ell]$ of size at most $\lambda_{k\mathcal{T}}(t) + |F(V_r)|$. The assumption $\lambda_{k\mathcal{T}}(t) < \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$ guarantees that $\lambda_{k\mathcal{T}}(t) + |F(V_r)| < \lambda_{k\mathcal{T}}(t_\ell)$, contradicting the minimality of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$. Using similar arguments we can show that $\lambda_{k\mathcal{T}}(t) \ge \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|$.

We now show that $\lambda_{k\mathcal{T}}(t) \ge \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k}\Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu}\Upsilon_\ell^i$. Assume that $\lambda_{k\mathcal{T}}(t) < \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k}\Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu}\Upsilon_\ell^i$. Consider removing from this path cover all the free vertices. What results is a set of paths which is clearly a kFSPC for $G[t]$. Since the removal of a free vertex in will increase the number of paths by at most one, we obtain a kFSPC of $G[t]$ of size at most $\lambda_{k\mathcal{T}}(t) + |F(V_\ell)| + |F(V_r)|$. The assumption $\lambda_{k\mathcal{T}}(t) < \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k} \Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu} \Upsilon_\ell^i$ guarantees that $\lambda_{k\mathcal{T}}(t) + |F(V_\ell)| + |F(V_r)| < \Sigma_{i=1}^{\nu}\lambda_\ell^i + \sum_{i=\nu+1}^{k} \Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i + \sum_{i=1}^{\nu} \Upsilon_\ell^i \leq \Sigma_{i=1}^{\nu}\lambda_\ell^i + \Sigma_{i=\nu+1}^{k}|T_r'^i| + \Sigma_{i=\nu+1}^{k}\lambda_r^i + \Sigma_{i=1}^{\nu}|T_\ell'^i|$, where $T_r'^i$ denotes the set of the terminal paths of T_r^i containing at least one free vertex. However, if $G[t_\ell]$ and $G[t_r]$ contain no free vertices then $\lambda_{k\mathcal{T}}(t) \geq \Sigma_{i=1}^{\nu}|T_\ell^i| + \Sigma_{i=1}^{\nu}|S_\ell^i| + \Sigma_{i=1}^{\nu}|T_\ell'^i| + \Sigma_{i=\nu+1}^{k}|T_r^i| + \Sigma_{i=\nu+1}^{k}|S_r^i| + \Sigma_{i=\nu+1}^{k}|T_r'^i| = \Sigma_{i=1}^{\nu}\lambda_\ell^i + \Sigma_{i=\nu+1}^{k}|T_r'^i| + \Sigma_{i=\nu+1}^{k}\lambda_r^i + \Sigma_{i=1}^{\nu}|T_\ell'^i|$, a contradiction. Hence, the lemma follows. ▮

According to Lemma 6.2 the size $\lambda_{k\mathcal{T}}(t)$ of a minimum kFSPC of the graph $G[t]$ is greater than or equal to the size $\lambda'_{k\mathcal{T}}(t)$ of the kFSPC constructed by our algorithm. Thus, the size of a minimum kFSPC of the graph $G[t]$ is equal to $\lambda'_{k\mathcal{T}}(t)$ and, consequently, the kFSPC constructed by our algorithm is a minimum kFSPC of $G[t]$. Thus, we can state the following result.

Lemma 6.3. *Subroutine process(t) returns a minimum kFSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of the graph $G[t]$, for every internal S-node $t \in T(G)$.*

Since the above result holds for every S-node t of the modified co-tree $T(G)$, it also holds when t is the root of $T(G)$ and $\mathcal{T}_t^i = \mathcal{T}^i$, $i \in [1,k]$. Thus, the following theorem holds:

Theorem 6.1. *Let G be a cograph and let $\mathcal{T}^i$, $i \in [1,k]$ be k disjoint subsets of $V(G)$. Let t be the root of the modified co-tree $T(G)$, and let $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ be a minimum kFSPC of $G[t_\ell]$ and $G[t_r]$, respectively. Algorithm Minimum_kFSPC correctly computes a minimum kFSPC of $G = G[t]$ with respect to $\mathcal{T}^i = \mathcal{T}_t^i$, $i \in [1,k]$, of size $\lambda_{k\mathcal{T}} = \lambda_{k\mathcal{T}}(t)$, where*

$$\lambda_{k\mathcal{T}}(t) = \begin{cases} \lambda_{k\mathcal{T}}(t_r) + \lambda_{k\mathcal{T}}(t_\ell) & \textit{if } t \textit{ is a P-node,} \\ & \\ \max\{\lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|,\ \Sigma_{i=1}^{k}\lceil\frac{|\mathcal{T}^i|}{2}\rceil, & \\ \Sigma_{i=1}^{\nu}\lambda_\ell^i - |F(V_r)| + \sum_{i=\nu+1}^{k}\Upsilon_r^i + \Sigma_{i=\nu+1}^{k}\lambda_r^i - |F(V_\ell)| + \sum_{i=1}^{\nu}\Upsilon_\ell^i\} & \textit{if } t \textit{ is an S-node} \end{cases}$$

where $\mathcal{T}^i$, $1 \leq i \leq \nu$ (resp. $\nu + 1 \leq i \leq k$) denotes the terminal set $\mathcal{T}^i$ for which $|\mathcal{T}_\ell^i| \geq |\mathcal{T}_r^i|$ (resp. $|\mathcal{T}_\ell^i| < |\mathcal{T}_r^i|$) and $\sum_{i=\nu+1}^{k}\Upsilon_r^i$ (resp. $\sum_{i=1}^{\nu}\Upsilon_\ell^i$) denotes the number of paths of T_r^i (resp. T_ℓ^i) that Algorithm Minimum_kFSPC breaks into semi-terminal paths.

Let G be a cograph on n vertices and m edges, $\mathcal{T}^i$, $i \in [1,k]$ be k terminal sets, and let t be an S-node of the modified co-tree $T(G)$. From the description of the algorithm we can easily conclude that a minimum kFSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of $G[t]$ can be constructed in $O(E(G[t]))$ time, since we use at most $|V(G[t_\ell])| \cdot |V(G[t_r])|$ edges to connect the paths of the minimum kFSPCs of the graphs $G[t_\ell]$ and $G[t_r]$; in the case where t is a P-node a minimum kFSPC is constructed in $O(1)$ time. Thus, the time needed by the subroutine process(t) to compute a minimum kFSPC in the case where t is the root of the tree $T(G)$ is $O(n+m)$; moreover, through the execution of the subroutine no additional space is needed. The construction of the co-tree $T_{co}(G)$ of G needs $O(n+m)$ time and it requires $O(n)$ space [22, 24]. Furthermore, the binarization process of the co-tree, that is, the construction of the modified co-tree $T(G)$, takes $O(n)$ time. Hence, we can state the following result.

Theorem 6.2. *Let G be a cograph on n vertices and m edges and let $\mathcal{T}^i$, $i \in [1,k]$, be k disjoint subsets of $V(G)$. A minimum k-fixed-set path cover $\mathcal{P}_{k\mathcal{T}}$ of G can be computed in $O(n+m)$ time and space.*

6.5 Concluding Remarks

In this chapter, we introduce the k-fixed-set path cover problem (kFSPC) and propose a polynomial-time solution on the class of cographs. Our algorithm produces a minimum k-fixed-set path cover of a cograph G that contains the largest number of terminal paths. It would be interesting to see if the k-fixed-set path cover problem can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. This promises to be an interesting area for further research since the complexity status of simpler problems, such as the 2HP problem, is unknown for interval graphs.

Chapter 7

The 2-terminal-set Path Cover Problem is Polynomial on Cographs

7.1 Introduction

In this chapter, we study a generalization of the path cover problem, namely, the 2-terminal-set path cover problem, or 2TPC for short. Given a graph G and two disjoint subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of $V(G)$, a 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ are all endpoints of the paths in $\mathcal{P}$ and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$. Note that, if $\mathcal{T}^1 \cup \mathcal{T}^2$ is empty, the stated problem coincides with the classical path cover problem. The 2TPC problem generalizes some path cover related problems, such as the 1HP and 2HP problems, which have been proved to be NP-complete even for small classes of graphs, and also Problem B described in Section 5.1. We show that the 2TPC problem can be solved in linear time on the class of cographs. The proposed linear-time algorithm is simple, requires linear space, and also enables us to solve the 1HP and 2HP problems on cographs within the same time and space complexity. We next define the 2TPC problem.

Problem 2TPC. Let G be a graph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint sets of vertices of $V(G)$. A *2-terminal-set path cover* of the graph G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a path cover of G such that all vertices in $\mathcal{T}^1 \cup \mathcal{T}^2$ are endpoints of paths in the path cover and all the paths with both endpoints in

$\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$; a *minimum 2-terminal-set path cover* of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a 2-terminal-set path cover of G with minimum cardinality; the *2-terminal-set path cover problem* (2TPC) is to find a minimum 2-terminal-set path cover of the graph G.

Contribution. In this chapter, we show that the 2-terminal-set path cover problem (2TPC) has a polynomial-time solution in the class of complement reducible graphs, or cographs [24]. More precisely, we establish a lower bound on the size of a minimum 2-terminal-set path cover of a cograph G on n vertices and m edges. We then define path operations, and prove structural properties for the paths of such a path cover, which enable us to describe a simple algorithm for the 2TPC problem. The proposed algorithm runs in time linear in the size of the input graph G, that is, in $O(n + m)$ time, and requires linear space.

The proposed algorithm for the 2TPC problem can also be used to solve the 1HP and 2HP problems on cographs within the same time and space complexity. Moreover, we have designed our algorithm so that it produces a minimum 2-terminal-set path cover of a cograph G that contains a large number of paths with one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$ (we can easily find a graph G and two sets $\mathcal{T}^1$ and $\mathcal{T}^2$ of vertices of $V(G)$ so that G admits two minimum 2-terminal-set path covers with different numbers of paths having one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$; for example, consider the graph G with vertex set $V(G) = \{a, b, c, d\}$, edge set $E(G) = \{ab, bc, ac, cd\}$, and $\mathcal{T}^1 = \{a\}$, $\mathcal{T}^2 = \{b\}$).

Road Map. The chapter is organized as follows. In Section 7.2 we establish the notation and related terminology, and we present background results. In Section 7.3 we describe our linear-time algorithm for the 2TPC problem, while in Section 7.4 we prove its correctness and compute its time and space complexity. Finally, in Section 7.5 we conclude the chapter and discuss possible future extensions.

7.2 Theoretical Framework

For convenience and ease of presentation, we binarize the co-tree $T_{co}(G)$ of a cograph G in such a way that each of its internal nodes has exactly two children [61, 70]. We shall refer to the binarized version of $T_{co}(G)$ as the modified co-tree of G and will denote it by $T(G)$. Thus, the left and right child of an internal node t of $T(G)$ will be denoted by t_ℓ and t_r, respectively. Let t be an internal node of $T(G)$. Then $G[t]$ is the subgraph of G induced by the subset V_t of the vertex set $V(G)$, which contains all the vertices of G that have as common ancestor in $T(G)$ the node t. For simplicity, we will denote by V_ℓ and V_r the vertex sets $V(G[t_\ell])$ and $V(G[t_r])$, respectively.

Let G be a cograph, $\mathcal{T}^1$ and $\mathcal{T}^2$ be two sets of vertices of $V(G)$ such that $\mathcal{T}^1 \cap \mathcal{T}^2 = \emptyset$, and let $\mathcal{P}_{2\mathcal{T}}(G)$ be a minimum 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ of size $\lambda_{2\mathcal{T}}$; note that the size of $\mathcal{P}_{2\mathcal{T}}(G)$ is the number of paths it contains. The vertices of the sets $\mathcal{T}^1$ and $\mathcal{T}^2$ are called *terminal* vertices, and the sets $\mathcal{T}^1$ and $\mathcal{T}^2$ are called the *terminal sets* of G, while those of $V(G) - (\mathcal{T}^1 \cup \mathcal{T}^2)$ are called *non-terminal or free* vertices. Thus, the set $\mathcal{P}_{2\mathcal{T}}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices belonging to different sets, that is, $u \in \mathcal{T}^1$ and $v \in \mathcal{T}^2$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}^1$ or $\mathcal{T}^2$ and the other in $V(G)-(\mathcal{T}^1 \cup \mathcal{T}^2)$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}^1 \cup \mathcal{T}^2$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G) - (\mathcal{T}^1 \cup \mathcal{T}^2)$; if P_f consists of only one vertex, say, u, then $u \in V(G) - (\mathcal{T}^1 \cup \mathcal{T}^2)$.

The set of the non-terminal paths in a minimum 2TPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Furthermore, let S^1 and S^2 denote the sets of the semi-terminal paths such that the terminal vertices belong to $\mathcal{T}^1$ and $\mathcal{T}^2$, respectively. Thus, $|S| = |S^1| + |S^2|$ and the following equation holds.

$$\lambda_{2\mathcal{T}} = |N| + |S| + |T| = |N| + |S^1| + |S^2| + |T| \tag{7.1}$$

From the definition of the 2-terminal-set path cover problem (2TPC), we can easily conclude that the number of paths in a minimum 2TPC can not be less than the number of the terminal vertices of the terminal set having maximum cardinality. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T| = |S^1| + |S^2| + 2|T|$. Thus, we have the following proposition, which also holds for general graphs:

Proposition 7.1. *Let G be a cograph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. Then $|\mathcal{T}^1| = |S^1| + |T|$, $|\mathcal{T}^2| = |S^2| + |T|$ and $\lambda_{2\mathcal{T}} \geq \max\{|\mathcal{T}^1|, |\mathcal{T}^2|\}$.*

Clearly, the size of a 2TPC of a cograph G, as well as the size of a minimum 2TPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_{2\mathcal{T}} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G; hereafter, $F(V) = F(V(G))$. Furthermore, let $\mathcal{P}$ be a set of paths and let $V_{\mathcal{P}}$ denote the set of vertices belonging to the paths of the set $\mathcal{P}$; hereafter, $F(\mathcal{P}) = F(V_{\mathcal{P}})$. Then, if $\mathcal{T}^1$ and $\mathcal{T}^2$ are two disjoint subsets of $V(G)$, we have $\lambda_{2\mathcal{T}} \leq |F(V)| + |\mathcal{T}^1| + |\mathcal{T}^2|$.

Let t be an internal node of the tree $T(G)$, that is, t is either an S-node or a P-node [68]. Then $\lambda_{2\mathcal{T}}(t)$ denotes the number of paths in a minimum 2TPC of the graph $G[t]$ with respect to $\mathcal{T}_t^1$ and $\mathcal{T}_t^2$, where $\mathcal{T}_t^1$ and $\mathcal{T}_t^2$ are the terminal vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ of the graph $G[t]$, respectively. Let t_ℓ and t_r be the left and the right child of node t, respectively. We denote by $\mathcal{T}_\ell^1$ and $\mathcal{T}_r^1$ (resp. $\mathcal{T}_\ell^2$ and $\mathcal{T}_r^2$) the terminal vertices of $\mathcal{T}^1$ (resp. $\mathcal{T}^2$) in V_ℓ and V_r, respectively, where $V_\ell = V(G[t_\ell])$ and $V_r = V(G[t_r])$. Let N_ℓ, S_ℓ and T_ℓ be the sets of the non-terminal, semi-terminal and terminal paths in a minimum 2TPC of $G[t_\ell]$, respectively. Similarly, let N_r, S_r and T_r be the sets of the non-terminal, semi-terminal and terminal paths in a minimum 2TPC of $G[t_r]$, respectively. Note that S_ℓ^1 and S_r^1 (resp. S_ℓ^2 and S_r^2) denote the sets of the semi-terminal paths in a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively, containing a terminal vertex of $\mathcal{T}^1$ (resp. $\mathcal{T}^2$). Obviously, Eq. (7.1) holds for $G[t]$ as well, with t being either an S-node or a P-node, that is,

$$\lambda_{2\mathcal{T}}(t) = |N_t| + |S_t| + |T_t| = |N_t| + |S_t^1| + |S_t^2| + |T_t| \tag{7.2}$$

where N_t, S_t and T_t are the sets of the non-terminal, the semi-terminal and the terminal paths, respectively, in a minimum 2TPC of $G[t]$, that is in $\mathcal{P}_{2\mathcal{T}}(t)$, and S_t^1 and S_t^2 denote the sets of the semi-terminal paths in $\mathcal{P}_{2\mathcal{T}}(t)$ containing a terminal vertex of $\mathcal{T}^1$ and $\mathcal{T}^2$, respectively. If t is a P-node, then $\mathcal{P}_{2\mathcal{T}}(t) = \mathcal{P}_{2\mathcal{T}}(t_\ell) \cup \mathcal{P}_{2\mathcal{T}}(t_r)$, where $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ are minimum 2TPCs corresponding to $G[t_\ell]$ and $G[t_r]$, respectively, and $\lambda_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) + \lambda_{2\mathcal{T}}(t_r)$. Furthermore, in the case where t is a P-node, we have

$$\begin{array}{lclcl} |N_t| & = & |N_\ell| + |N_r| & & \\ |S_t| & = & |S_\ell| + |S_r| & = & |S_\ell^1| + |S_\ell^2| + |S_r^1| + |S_r^2| \\ |T_t| & = & |T_\ell| + |T_r| & & \end{array}$$

Thus, we focus on computing a minimum 2TPC of the graph $G[t]$ for the case where t is an S-node. Before describing our algorithm, we establish a lower bound on the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of a graph $G[t]$. More precisely, we prove the following lemma.

Lemma 7.1. *Let t be an internal node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. Then $\lambda_{2\mathcal{T}}(t) \geq \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\},\ \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Proof. Clearly, according to Proposition 7.1 and since $G[t]$ is a cograph, we have $\lambda_{2\mathcal{T}}(t) \geq \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$. We will prove that $\lambda_{2\mathcal{T}}(t) \geq \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$. Assume that $\lambda_{2\mathcal{T}}(t) < \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$. Consider removing from this path cover all the vertices in V_r. What results is a set of paths which is clearly a 2TPC for $G[t_\ell]$. Since the removal of a free vertex in $F(V_r)$ will increase the number of paths by at most one, we obtain a 2TPC of $G[t_\ell]$ of size at most $\lambda_{2\mathcal{T}}(t) + |F(V_r)|$. The assumption $\lambda_{2\mathcal{T}}(t) < \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$ guarantees that $\lambda_{2\mathcal{T}}(t) + |F(V_r)| < \lambda_{2\mathcal{T}}(t_\ell)$, contradicting the minimality of $\mathcal{P}_{2\mathcal{T}}(t_\ell)$. Using similar arguments we can show that $\lambda_{2\mathcal{T}}(t) \geq \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$. Hence, the lemma follows. ▮

We next define four operations on paths of a minimum 2TPC of the graphs $G[t_\ell]$ and $G[t_r]$, namely *break*, *connect*, *bridge* and *insert* operations; these operations are illustrated in Fig. 7.1.

- *Break* operation: Let $P = [p_1, p_2, \ldots, p_k]$ be a path of $\mathcal{P}_{2\mathcal{T}}(t_r)$ or $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ of length k. We say that we *break* the path P in two paths, say, P_1 and P_2, if we delete an arbitrary edge of P, say the edge p_ip_{i+1} $(1 \leq i < k)$, in order to obtain two paths which are $P_1 = [p_1, \ldots, p_i]$ and $P_2 = [p_{i+1}, \ldots, p_k]$. Note that we can break the path P in at most k trivial paths.

- *Connect* operation: Let P_1 be a non-terminal or a semi-terminal path of $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{2\mathcal{T}}(t_r)$) and let P_2 be a non-terminal or a semi-terminal path of $\mathcal{P}_{2\mathcal{T}}(t_r)$ (resp. $\mathcal{P}_{2\mathcal{T}}(t_\ell)$). We say that we *connect* the path P_1 with the path P_2, if we add an edge which joins two free endpoints of the two paths. Note that if $P_1 \in S_\ell^1$ (resp. $P_1 \in S_r^1$) then, if P_2 is also a semi-terminal path, $P_2 \in S_r^2$ (resp. $P_2 \in S_\ell^2$). Similarly, if $P_1 \in S_\ell^2$ (resp. $P_1 \in S_r^2$) then, if P_2 is also a semi-terminal path, $P_2 \in S_r^1$ (resp. $P_2 \in S_\ell^1$).

- *Bridge* operation: Let P_1 and P_2 be two paths of the set $N_\ell \cup S_\ell^1 \cup S_\ell^2$ (resp. $N_r \cup S_r^1 \cup S_r^2$) and let P_3 be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *bridge* the two paths P_1 and P_2 using path P_3 if we connect a free endpoint of P_1 with one endpoint of P_3 and a free endpoint of P_2 with the other endpoint of P_3. The result is a path having both endpoints in $G[t_\ell]$ (resp. $G[t_r]$). Note that if $P_1 \in S_\ell^1$ (resp. $P_1 \in S_r^1$) then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell^2$ (resp. $P_2 \in S_r^2$). Similarly, if $P_1 \in S_\ell^2$ (resp. $P_1 \in S_r^2$) then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell^1$ (resp. $P_2 \in S_r^1$).

- *Insert* operation: Let $P_1 = [t_1, p_1, \ldots, p_1', t_1']$ be a terminal path of the set T_ℓ (resp. T_r) and let $P_2 = [p_2, \ldots, p_2']$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *insert* the path P_2 into P_1, if we replace the first edge of P_1, that is, the edge t_1p_1, with the path $[t_1, p_2, \ldots, p_2', p_1]$. Thus, the resulting path is $P_1 = [t_1, p_2, \ldots, p_2', p_1, \ldots, p_1', t_1']$. Note that we can replace every edge of the terminal path so that we can insert at most $|F(\{P_1\})| + 1$ non-terminal paths, where $F(\{P_1\})$ is the set of the free vertices belonging to the path P_1. If the terminal path $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_1^r, \ldots, p_1', t_1']$ is constructed by connecting a semi-terminal path of S_ℓ, say, $P_\ell = [t_1, p_1, \ldots, p_1^\ell]$ with a semi-terminal path of S_r, say, $P_r = [p_1^r, \ldots, p_1', t_1']$, then it obviously has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. In this case, if $P_2 \in N_\ell$ (resp. N_r) we can only replace the edges of P_1 that belong to $G[t_r]$ (resp. $G[t_\ell]$). On the other hand, if P_2 has one endpoint, say, p_2, in N_ℓ and the other, say, p_2', in N_r, we insert P_2 into P_1 as follows: $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_2', \ldots, p_2, p_1^r, \ldots, p_1', t_1']$.

We can also combine the operations connect and bridge to perform a new operation which we call a *connect-bridge* operation; such an operation is depicted in Fig. 7.1(e) and is defined below.

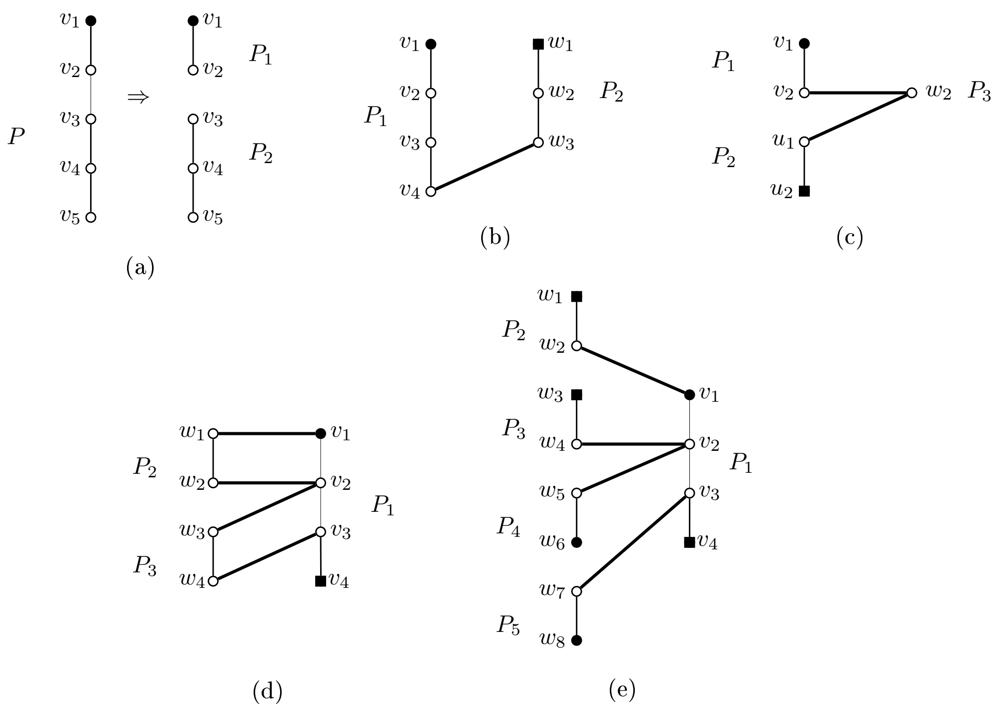


Figure 7.1: Illustrating (a) break, (b) connect, (c) bridge, (d) insert, and (e) connect-bridge operations; the vertices of $\mathcal{T}^1$ are denoted by black-circles, while the vertices of $\mathcal{T}^2$ are denoted by black-squares.

- *Connect-Bridge* operation: Let $P_1 = [t_1, p_1, \ldots, p_k, t_1']$ be a terminal path of the set T_ℓ (resp. T_r), where $t_1 \in \mathcal{T}^2$ and $t_1' \in \mathcal{T}^1$, and let $P_2, P_3, \ldots, P_{\frac{s+1}{2}}$ be semi-terminal paths of the set S_r^1 (resp. S_ℓ^1) and $P_{\frac{s+1}{2}+1}, \ldots, P_s$ be semi-terminal paths of the set S_r^2 (resp. S_ℓ^2), where s is odd and $3 \leq s \leq 2k+3$. We say that we *connect-bridge* the paths $P_2, P_3, \ldots, P_s$ using vertices of P_1, if we perform the following operations: (i) connect the path P_2 with the path $[t_1]$; (ii) bridge $r = \frac{s-3}{2}$ pairs of different semi-terminal paths using vertices $p_1, p_2, \ldots, p_r$; and (iii) connect the path $[p_{r+1}, \ldots, p_k, t_1']$ with the last semi-terminal path P_s.

The Connect-Bridge operation produces two paths having one endpoint in $G[t_\ell]$ and the other endpoint in $G[t_r]$ and $\frac{s-3}{2}$ paths having both endpoints in $G[t_r]$ (resp. $G[t_\ell]$).

7.3 The Algorithm

We next present an optimal algorithm for the 2TPC problem on cographs. Our algorithm takes as input a cograph G and two subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of its vertices, where $\mathcal{T}^1 \cap \mathcal{T}^2 = \emptyset$, and finds the paths of a minimum 2TPC of G in linear time; it works as follows:

Algorithm Minimum_2TPC

Input: a cograph G and two subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of $V(G)$ such that $\mathcal{T}^1 \cap \mathcal{T}^2 = \emptyset$;

Output: a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(G)$ of the cograph G;

1. Construct the co-tree $T_{co}(G)$ of G and make it binary; let $T(G)$ be the resulting tree;
2. Execute the subroutine $process(root)$, where $root$ is the root node of the tree $T(G)$; the minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(root) = \mathcal{P}_{2\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

Algorithm 5: Algorithm Minimum_2TPC

where the description of the subroutine *process*( ) is as follows:

process (node t)

Input: node t of the modified co-tree $T(G)$ of the input graph G.

Output: a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of the cograph $G[t]$.

```
1. if $t$ is a leaf
   then return($\{u\}$), where $u$ is the vertex associated with the leaf $t$;
   else {$t$ is an internal node that has a left and a right child denoted by $t_\ell$ and $t_r$, resp.}
         $\mathcal{P}_{2\mathcal{T}}(t_\ell) \leftarrow process(t_\ell)$;
         $\mathcal{P}_{2\mathcal{T}}(t_r) \leftarrow process(t_r)$;

2. if $t$ is a P-node
   then return($\mathcal{P}_{2\mathcal{T}}(t_\ell) \cup \mathcal{P}_{2\mathcal{T}}(t_r)$);

3. if $t$ is an S-node
   then if $|N_\ell| \le |N_r|$ then swap($\mathcal{P}_{2\mathcal{T}}(t_\ell), \mathcal{P}_{2\mathcal{T}}(t_r)$);
         $s^1 = |S^1_\ell| - |S^2_r|$;
         $s^2 = |S^2_\ell| - |S^1_r|$;
         case 1: $s^1 \ge 0$ and $s^2 \ge 0$
               call procedure $2TPC\_1$;
         case 2: $s^1 < 0$ and $s^2 < 0$
               if $|N_r| + \min\{|s^1|, |s^2|\} \le |F(S^1_\ell \cup S^2_\ell \cup N_\ell)|$
               then call procedure $2TPC\_2\_a$;
               else call procedure $2TPC\_2\_b$;
         case 3: ($s^1 \ge 0$ and $s^2 < 0$) or ($s^1 < 0$ and $s^2 \ge 0$)
               call procedure $2TPC\_3$;
```

We next describe the subroutine *process*( ) in the case where t is an S-node of $T(G)$. Note that, if $|N_\ell| \le |N_r|$, we swap $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$. Thus, we distinguish the following three cases: (1) $s^1 \ge 0$ and $s^2 \ge 0$, (2) $s^1 < 0$ and $s^2 < 0$, and (3) ($s^1 \ge 0$ and $s^2 < 0$) or ($s^1 < 0$ and $s^2 \ge 0$).

Case 1: $s^1 \ge 0$ and $s^2 \ge 0$

Let SN_r be the set of non-terminal paths obtained by breaking the set $S^1_r \cup S^2_r \cup N_r$ into $|N_\ell| - 1 + \min\{s^1, s^2\}$ non-terminal paths; thus, $|SN_r| \le |F(S^1_r \cup S^2_r \cup N_r)|$. In the case where $|N_\ell| - 1 + \min\{s^1, s^2\} \ge F(S^1_r \cup S^2_r \cup N_r)$, the paths of SN_r are trivial (recall that $F(S^1_r \cup S^2_r \cup N_r)$ is the set of free vertices belonging to the set $S^1_r \cup S^2_r \cup N_r$). The paths of SN_r are used to bridge at most $2\min\{s^1, s^2\}$ semi-terminal paths of $S^1_\ell \cup S^2_\ell$ and, if $|SN_r| - \min\{s^1, s^2\} > 0$, at most $|N_\ell|$ non-terminal paths of N_ℓ. We can construct the paths of a 2TPC using the following procedure:

Procedure 2TPC_1

1. connect the $|S^2_r|$ paths of S^2_r with $|S^2_r|$ paths of S^1_ℓ, and the $|S^1_r|$ paths of S^1_r with $|S^1_r|$ paths of S^2_ℓ;
2. bridge $2\min\{s^1, s^2\}$ semi-terminal paths of $S^1_\ell \cup S^2_\ell$ using $\min\{s^1, s^2\}$ paths of SN_r;
3. bridge the non-terminal paths of N_ℓ using $|N_\ell| - 1$ non-terminal paths of SN_r; this produces non-terminal paths with both endpoints in $G[t_\ell]$, unless $|N_\ell| \leq |F(S^1_r \cup S^2_r \cup N_r)| - \min\{s^1, s^2\}$ where we obtain one non-terminal path with one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;
4. if $|N_\ell| \leq |F(S^1_r \cup S^2_r \cup N_r)| - \min\{s^1, s^2\}$ insert the non-terminal path obtained in Step 3 into one terminal path which is obtained in Step 1;
5. if $|T_r| = |S^1_\ell| = |S^2_\ell| = 0$ and $|F(S^1_r \cup S^2_r \cup N_r)| \geq |N_\ell| + 1$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and insert it into a terminal path of T_ℓ;
6. if $|T_r| = |S^1_r| = |S^2_r| = 0$ and $|F(N_r)| \geq |N_\ell| + \min\{s^1, s^2\}$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and use it to connect two semi-terminal paths of $S^1_\ell \cup S^2_\ell$;
7. if $s^1 - \min\{\min\{s^1, s^2\},\ |F(S^1_r \cup S^2_r \cup N_r)|\}$ (resp. $s^2 - \min\{\min\{s^1, s^2\},\ |F(S^1_r \cup S^2_r \cup N_r)|\}$) is odd and there is at least one free vertex in $S^1_r \cup S^2_r \cup N_r$ which is not used in Steps 1–6, or there is a non-terminal path having one endpoint in $G[t_\ell]$ and the other in $G[t_r]$, connect one non-terminal path with one semi-terminal path of S^1_ℓ (resp. S^2_ℓ);
8. connect-bridge the rest of the semi-terminal paths of $S^1_\ell \cup S^2_\ell$ (at most $2(|F(T_r)| + |T_r|)$) using vertices of T_r;
9. insert non-terminal paths obtained in Step 3 into the terminal paths of T_r;

Based on the procedure 2TPC_1, we can compute the cardinality of the sets N_t, S^1_t, S^2_t and T_t, and thus, since $\lambda'_{2T}(t) = |N_t| + |S_t| + |T_t|$ and $|S_t| = S^1_t + S^2_t$, the number of paths in the 2TPC constructed by the procedure at node $t \in T(G)$. In this case, the values of $|N_t|$, $|S_t|$ and $|T_t|$ are the following:

$$\begin{aligned}
|N_t| &= \max\{\mu - \alpha,\ 0\} \\
|S^1_t| &= \min\{\sigma^1_\ell,\ \max\{\sigma^1_\ell - |F(T_r)| - |T_r|,\ \max\{\sigma^1_\ell - \sigma^2_\ell, 0\}\}\} \\
|S^2_t| &= \min\{\sigma^2_\ell,\ \max\{\sigma^2_\ell - |F(T_r)| - |T_r|,\ \max\{\sigma^2_\ell - \sigma^1_\ell, 0\}\}\} \\
|S_t| &= |S^1_t| + |S^2_t| \\
|T_t| &= |S^1_r| + |S^2_r| + \min\{\min\{s^1,\ s^2\},\ |F(S^1_r \cup S^2_r \cup N_r)|\} + |T_\ell| + |T_r| + \tfrac{\sigma^1_\ell + \sigma^2_\ell - |S_t|}{2}
\end{aligned} \tag{7.3}$$

where

$$\begin{aligned}
\sigma^1_\ell &= |S^1_\ell| - |S^2_r| - \min\{\min\{s^1,\ s^2\},\ |F(S^1_r \cup S^2_r \cup N_r)|\}, \\
\sigma^2_\ell &= |S^2_\ell| - |S^1_r| - \min\{\min\{s^1,\ s^2\},\ |F(S^1_r \cup S^2_r \cup N_r)|\}, \\
\mu &= \max\{|N_\ell| - \pi_r, \max\{1 - \max\{|S^1_\ell|, |S^2_\ell|\}, 0\}\} - \max\{|F(T_r)| + |T_r| - \min\{\sigma^1_\ell, \sigma^2_\ell\}, 0\} - \\
&\quad \min\{\max\{\min\{|N_\ell| - \pi_r, \delta(\sigma^1_\ell), \delta(\sigma^2_\ell)\}, 0\}, \max\{\min\{F(S^1_r \cup S^2_r \cup N_r) - \min\{s^1, s^2\}, 1\}, 0\}\}, \\
\alpha &= \min\{\max\{\min\{\pi_r - |N_\ell|,\ 1\},\ 0\},\ \max\{|T_\ell|,\ 0\}\}, \text{ and} \\
\pi_r &= \max\{|F(S^1_r \cup S^2_r \cup N_r)| - \min\{s^1,\ s^2\},\ 0\}.
\end{aligned}$$

In Eq. (7.3), σ_ℓ^1 (resp. σ_ℓ^2) is the number of semi-terminal paths of S_ℓ^1 (resp. S_ℓ^2) that are not connected or bridged at Steps 1–3. Furthermore, π_r is the number of free vertices in the set $S_r^1 \cup S_r^2 \cup N_r$ that are not used to bridge semi-terminal paths of $S_\ell^1 \cup S_\ell^2$ at Step 3 and δ is a function which is defined as follows: $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise. Note that at most $|F(T_r)| + |T_r|$ non-terminal paths can be inserted into the terminal paths of T_r or the terminal paths can connect-bridge at most $2(|F(T_r)| + |T_r|)$ semi-terminal paths.

Case 2: $s^1 < 0$ and $s^2 < 0$

In this case, we need $|N_r| + \min\{|s^1|, |s^2|\}$ paths of $G[t_\ell]$ in order to bridge $|N_r|$ non-terminal paths of N_r and $2\min\{|s^1|, |s^2|\}$ semi-terminal paths of $S_r^1 \cup S_r^2$. If $|N_\ell| < |N_r| + \min\{|s^1|, |s^2|\}$ we break the non-terminal paths of N_ℓ into at most $|F(N_\ell)|$ paths; in the case where $|F(N_\ell)| < |N_r| + \min\{|s^1|, |s^2|\}$ we also use (at most $|F(S_\ell^1 \cup S_\ell^2)|$) vertices of $S_\ell^1 \cup S_\ell^2$. Let $p = \min\{|N_r| + \min\{|s^1|, |s^2|\},\ |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|\}$. We distinguish two cases:

2.a $|N_r| + \min\{|s^1|, |s^2|\} \leq |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$.

In this case, $p = |N_r| + \min\{|s^1|, |s^2|\}$ and the number of non-terminal paths (or free vertices) of $G[t_\ell]$ is sufficient to bridge non-terminal paths of N_r and semi-terminal paths of $S_r^1 \cup S_r^2$. In detail, let SN_ℓ be the set of non-terminal paths obtained by breaking the set $S_\ell^1 \cup S_\ell^2 \cup N_\ell$ into p non-terminal paths in order to bridge $2\min\{|s^1|, |s^2|\}$ semi-terminal paths of $S_r^1 \cup S_r^2$ and all the non-terminal paths of N_r. If $p < |N_\ell|$ then $SN_\ell = N_\ell$. Obviously, $|SN_\ell| \leq |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$. Note that, if $p < |N_\ell|$ then the non-terminal paths of N_r are used to bridge the paths of N_ℓ. More precisely, we use paths of the set SN_r (it is the set of non-terminal paths that we get by breaking the set $S_r^1 \cup S_r^2 \cup N_r$) in order to obtain $|N_\ell| - \min\{|s^1|, |s^2|\}$ non-terminal paths. If $p \geq |N_\ell|$ we set $SN_r = N_r$ and we use at most $|F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$ paths obtained by $S_\ell^1 \cup S_\ell^2 \cup N_\ell$ in order to bridge non-terminal paths of N_r and semi-terminal paths of $S_r^1 \cup S_r^2$, that is, we use the set SN_ℓ. As a result, we construct $\min\{|s^1|, |s^2|\}$ terminal paths having both of their endpoints in $G[t_r]$ and we have at least one non-terminal path, if $p < |N_\ell|$, and exactly one non-terminal path, otherwise. Note that, in the second case, we can construct the non-terminal path in such a way that one endpoint is in SN_ℓ and the other is in N_r. We construct the paths of a 2TPC at node $t \in T(G)$ using the following procedure:

Procedure 2TPC_2_a

1. connect the $|S_\ell^1|$ paths of S_ℓ^1 with $|S_\ell^1|$ paths of S_r^2, and the $|S_\ell^2|$ paths of S_ℓ^2 with $|S_\ell^2|$ paths of S_r^1;
2. if $|T_\ell| = |T_r| = 0$ and $p \geq |N_\ell|$, use N_r to bridge $p - \min\{|s^1|, |s^2|\} + 1$ paths of SN_ℓ and use the constructed non-terminal path having both of its endpoints in $G[t_\ell]$ to bridge two semi-terminal paths of $S_r^1 \cup S_r^2$;
3. bridge semi-terminal paths of $S_r^1 \cup S_r^2$ using paths of SN_ℓ;
4. if $|T_r| = 0$, $|T_\ell| \neq 0$, $p \geq |N_\ell|$ and $|F(S_r^1 \cup S_r^2 \cup N_r)| \geq |SN_\ell| - \min\{|s^1|, |s^2|\}$ construct a non-terminal path having both of its endpoints in $G[t_r]$ and use a terminal path of T_ℓ to insert the constructed non-terminal path;
5. bridge the remaining paths of SN_ℓ using the paths of SN_r. This produces non-terminal paths one of which has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;
6. if $|s^2| - \min\{|s^1|, |s^2|\}$ (resp. $|s^1| - \min\{|s^1|, |s^2|\}$) is odd, we connect one non-terminal path with one semi-terminal path of S_r^1 (resp. S_r^2);
7. insert at most $|F(T_r)| + |T_r|$ non-terminal paths obtained in Step 5 into the terminal paths of T_r;

Based on the path operations performed by procedure 2TPC_2_a, we can compute the cardinalities of the sets N_t, S_t and T_t:

$$\begin{array}{rcl} |N_t| &=& \max\{\mu - \alpha,\ 0\} \\ |S_t^1| &=& \max\{|s^2| - |s^1|,\ 0\} \\ |S_t^2| &=& \max\{|s^1| - |s^2|,\ 0\} \\ |S_t| &=& |S_t^1| + |S_t^2| \\ |T_t| &=& |S_\ell^1| + |S_\ell^2| + \min\{|s^1|,\ |s^2|\} + |T_\ell| + |T_r| \end{array} \tag{7.4}$$

where

$$\begin{array}{rcl} \mu &=& \max\{|N_\ell| - F(S_r^1 \cup S_r^2 \cup N_r),\ 0\} - \min\{|s^1|,\ |s^2|\} - |F(T_r)| - |T_r| - \\ && \max\{\delta(|s^1| - \min\{|s^1|,\ |s^2|\}), \delta(|s^2| - \min\{|s^1|,\ |s^2|\})\},\ \text{and} \\ \alpha &=& \min\{\max\{\min\{F(S_r^1 \cup S_r^2 \cup N_r) - |N_\ell|,\ 1\},\ 0\},\ \max\{|T_\ell|,\ 0\}\}. \end{array}$$

2.b $|N_r| + \min\{|s^1|, |s^2|\} > |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$.

In this case, $p = |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$ and the number of free vertices of $G[t_\ell]$ (that is, in $S_\ell^1 \cup S_\ell^2 \cup N_\ell$) is not sufficient to bridge non-terminal paths of N_r and semi-terminal paths of $S_r^1 \cup S_r^2$. In detail, let SN_ℓ be the set of the trivial, non-terminal paths, obtained by breaking the set $S_\ell^1 \cup S_\ell^2 \cup N_\ell$ into $|F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|$ non-terminal paths. We can construct the paths of a 2TPC at node $t \in T(G)$ using the following procedure:

Procedure 2TPC_2_b

1. connect the $|S_\ell^1|$ paths of S_ℓ^1 with $|S_\ell^1|$ paths of S_r^2, and the $|S_\ell^2|$ paths of S_ℓ^2 with $|S_\ell^2|$ paths of S_r^1;
2. bridge $2\min\{|s^1|, |s^2|\}$ semi-terminal paths of $S_r^1 \cup S_r^2$ using $\min\{|s^1|, |s^2|\}$ paths of SN_ℓ;
3. bridge the non-terminal paths of N_r using the rest of the non-terminal paths of SN_ℓ. This produces non-terminal paths such that both endpoints belong to $G[t_r]$;
4. connect-bridge the rest of the semi-terminal paths of $S_r^1 \cup S_r^2$ (at most $2(|F(T_\ell)| + |T_\ell|)$) using vertices of T_ℓ;
5. insert non-terminal paths obtained in Step 3 into the terminal paths of T_ℓ;

Based on the procedure 2TPC_2_b, we can compute the cardinalities of non-terminal, semi-terminal and terminal sets:

$$\begin{array}{rcl} |N_t| &=& \max\{\mu,\ 0\} \\ |S_t^1| &=& \min\{\sigma_r^1,\ \max\{\sigma_r^1 - |F(T_\ell)| - |T_\ell|,\ \max\{\sigma_r^1 - \sigma_r^2, 0\}\}\} \\ |S_t^2| &=& \min\{\sigma_r^2,\ \max\{\sigma_r^2 - |F(T_\ell)| - |T_\ell|,\ \max\{\sigma_r^2 - \sigma_r^1, 0\}\}\} \\ |S_t| &=& |S_t^1| + |S_t^2| \\ |T_t| &=& |S_\ell^1| + |S_\ell^2| + \min\{\min\{|s^1|,\ |s^2|\},\ |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|\} + |T_\ell| + |T_r| + \frac{\sigma_r^1 + \sigma_r^2 - |S_t|}{2} \end{array} \tag{7.5}$$

where

$$\begin{array}{rcl} \sigma_r^1 &=& |S_r^1| - |S_\ell^2| - \min\{\min\{|s^1|,\ |s^2|\},\ |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|\}, \\ \sigma_r^2 &=& |S_r^2| - |S_\ell^1| - \min\{\min\{|s^1|,\ |s^2|\},\ |F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)|\}, \\ \mu &=& |N_r| - \pi_\ell - \min\{\max\{\min\{|F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)| - \min\{|s^1|,\ |s^2|\},\ 1\},\ 0\}, \\ && \max\{\min\{|N_r| - \pi_\ell,\ \delta(|s^1|),\ \delta(|s^2|)\},\ 0\}\} - \max\{|F(T_\ell)| + |T_\ell| - \min\{\sigma_r^1, \sigma_r^2\},\ 0\},\ \text{and} \\ \pi_\ell &=& \max\{|F(S_\ell^1 \cup S_\ell^2 \cup N_\ell)| - \min\{|s^1|,\ |s^2|\},\ 0\}. \end{array}$$

In Eq. (7.5), σ_r^1 (resp. σ_r^2) is the number of semi-terminal paths of S_r^1 (resp. S_r^2) that are not connected or bridged at Steps 1–3. Moreover, π_ℓ is the number of free vertices that belong to the set $S_\ell^1 \cup S_\ell^2 \cup N_\ell$ and are not used to bridge semi-terminal paths of $S_r^1 \cup S_r^2$ (at Step 3). Again, $\delta(x) = 1$, if x is odd, and $\delta(x) = 0$ otherwise. Note that at most $|F(T_\ell)| + |T_\ell|$ non-terminal paths can be inserted into the terminal paths of T_ℓ or the terminal paths can connect-bridge at most $2(|F(T_\ell)| + |T_\ell|)$ semi-terminal paths.

Case 3: ($s^1 \geq 0$ and $s^2 < 0$) or ($s^1 < 0$ and $s^2 \geq 0$)

Let SN_r be the set of non-terminal paths which are used to bridge at most $|N_\ell|$ non-terminal paths of N_ℓ; it is obtained by breaking the set $S_r^1 \cup S_r^2 \cup N_r$ into $|N_\ell| - 1$ non-terminal paths. In the case where $|N_\ell| - 1 \geq F(S_r^1 \cup S_r^2 \cup N_r)$, the paths of SN_r are trivial. We can construct the paths of a 2TPC using the following procedure:

Procedure 2TPC_3

1. connect $\min\{|S_\ell^1|, |S_r^2|\}$ paths of S_ℓ^1 with $\min\{|S_\ell^1|, |S_r^2|\}$ paths of S_r^2, and $\min\{|S_\ell^2|, |S_r^1|\}$ paths of S_ℓ^2 with $\min\{|S_\ell^2|, |S_r^1|\}$ paths of S_r^1;
2. bridge the non-terminal paths of N_ℓ using $|N_\ell| - 1$ non-terminal paths of SN_r; this produces non-terminal paths with both endpoints in $G[t_\ell]$, unless $|N_\ell| \leq |F(S_r^1 \cup S_r^2 \cup N_r)|$ where we obtain one non-terminal path with one endpoint in $G[t_\ell]$ and the other in $G[t_r]$;
3. if $|N_\ell| \leq |F(S_r^1 \cup S_r^2 \cup N_r)|$ insert the non-terminal path obtained in Step 2 into one terminal path which is obtained in Step 1;
4. if there is at least one free vertex in $S_r^1 \cup S_r^2 \cup N_r$ which is not used in Steps 1–3, or there is a non-terminal path having one endpoint in $G[t_\ell]$ and the other in $G[t_r]$, connect one non-terminal path with one semi-terminal path of $S_\ell^1 \cup S_\ell^2$;
5. if there is a non-terminal path having at least one endpoint in $G[t_\ell]$, connect it with one semi-terminal path of $S_r^1 \cup S_r^2$;
6. insert non-terminal paths obtained in Step 2 into the terminal paths of T_r;

Based on the procedure 2TPC_3, we can compute the values of $|N_t|$, $|S_t|$ and $|T_t|$:

$$\begin{array}{rcl}
|N_t| & = & \max\{|N_\ell| - |F(V_r)| - |T_r| - \max\{\sigma_r^1,\ \sigma_r^2\},\ 0\} \\
|S_t^1| & = & \sigma_\ell^1 + \sigma_r^1 \\
|S_t^2| & = & \sigma_\ell^2 + \sigma_r^2 \\
|S_t| & = & |S_t^1| + |S_t^2| \\
|T_t| & = & \min\{|S_\ell^1|,\ |S_r^2|\} + \min\{|S_\ell^2|,\ |S_r^1|\} + |T_\ell| + |T_r|
\end{array} \tag{7.6}$$

where

$$\begin{array}{rcl}
\sigma_\ell^1 & = & \max\{|S_\ell^1| - |S_r^2|,\ 0\}, \\
\sigma_\ell^2 & = & \max\{|S_\ell^2| - |S_r^1|,\ 0\}, \\
\sigma_r^1 & = & \max\{|S_r^1| - |S_\ell^2|,\ 0\}, \text{ and} \\
\sigma_r^2 & = & \max\{|S_r^2| - |S_\ell^1|,\ 0\}.
\end{array}$$

In Eq. (7.6), σ_ℓ^1 (resp. σ_ℓ^2) is the number of semi-terminal paths of S_ℓ^1 (resp. S_ℓ^2) that are not connected at Step 1 (resp. Step 2) and σ_r^1 (resp. σ_r^2) is the number of semi-terminal paths of S_r^1 (resp. S_r^2) that are not connected at Step 2 (resp. Step 1).

7.4 Correctness and Time Complexity

Let G be a cograph, $T(G)$ be the modified co-tree of G, and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be the two terminal sets of G. Since our algorithm computes a 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t)$ for each internal node $t \in T(G)$, and thus for the root $t = t_{root}$ of the tree $T(G)$, we need to prove that the constructed 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ is minimum. Obviously, the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC of the graph $G[t]$ is less than or equal to the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC constructed by our algorithm. According to Proposition 7.1, if the size of the 2TPC constructed by our algorithm is $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, then it is a minimum 2TPC. After performing simple computations we get four specific values for the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC constructed by our algorithm, that is, by the 2TPC procedures 1, 2_a, 2_b and 3. More precisely, if t is an internal S-node of $T(G)$, our algorithm returns a 2TPC of size $\lambda'_{2\mathcal{T}}(t)$ equal to either $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$, $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, $\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$, or $\lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$; see Table 7.1. Specifically, in the case where $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$ procedure 2TPC_1 returns a 2TPC of the graph $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$. We prove the following lemma, which shows that if the size of the 2TPC returned by our subroutine process(t) for the graph $G[t]$ is $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$ (procedure 2TPC_1), then it is a minimum 2TPC.

Lemma 7.2. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. If $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$, then the procedure 2TPC_1 returns a minimum 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$.*

Proof. Since we can construct a 2TPC of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$, then the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC is at most $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$. We will show that we can not construct a minimum 2TPC of size less than $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1$, that is, we will show that $\lambda_{2\mathcal{T}}(t) \geq \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}+1 \Leftrightarrow \lambda_{2\mathcal{T}}(t) > \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$. Thus, we only need to prove that $\lambda_{2\mathcal{T}}(t) \neq \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$. Note that by the assumption we have $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} = |T_\ell|$. We assume that $\lambda_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, and, thus, $\lambda_{2\mathcal{T}}(t) = |T_\ell|$. There exists at least one non-terminal path in $G[t_\ell]$; for otherwise $|N_\ell| = 0$, and thus $V_r = \emptyset$, a contradiction. We ignore the terminal paths from the minimum 2TPC of $G[t_\ell]$ and apply the algorithm described in [61] to $G[t]$. The resulting minimum 2TPC contains only one (non-terminal) path which either has both endpoints in $G[t_\ell]$ or it has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. This non-terminal path can not be inserted into a terminal path of $G[t_\ell]$ because it does not have both endpoints in $G[t_r]$. Thus, $\lambda_{2\mathcal{T}}(t) = |T_\ell| + 1$, a contradiction. ∎

Moreover, if the size of the 2TPC returned by the process(t) is $\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$ (all the procedures), then it is obviously a minimum 2TPC of $G[t]$. We prove that the size $\lambda'_{2\mathcal{T}}(t)$ of the 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ that our subroutine process(t) returns is minimum.

Lemma 7.3. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. If the subroutine process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, then $\lambda'_{2\mathcal{T}}(t) \geq \max\{\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Proof. Since $\lambda'_{2\mathcal{T}}(t) = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$, we have $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t)$, that is, the 2TPC that the subroutine process(t) returns is minimum. Thus, the proof follows from Lemma 7.1. ∎

Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. Furthermore, we assume that the conditions $|S_\ell^1| = |S_\ell^2| = |T_r| = |S_r^1| = |S_r^2| = 0$ and $|N_\ell| = |V_r|$ do not hold together. We consider the case where the subroutine process(t) returns a 2TPC $\mathcal{P}'_{2\mathcal{T}}(t)$ of the graph $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$ (cases 1, 2.a and 3). We prove the following lemma.

Procedures	Size of 2-terminal-set PC
Procedure 2TPC_1	$\max\{\lvert\mathcal{T}_t^1\rvert, \lvert\mathcal{T}_t^2\rvert\}+1$
All the procedures	$\max\{\lvert\mathcal{T}_t^1\rvert, \lvert\mathcal{T}_t^2\rvert\}$
Procedures 2TPC_1, 2TPC_2_a and 2TPC_3	$\lambda_{2\mathcal{T}}(t_\ell) - \lvert F(V_r)\rvert$
Procedure 2TPC_2_b	$\lambda_{2\mathcal{T}}(t_r) - \lvert F(V_\ell)\rvert$

Table 7.1: The size of the 2TPC that our algorithm returns in each case.

Lemma 7.4. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. If the subroutine process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$, then $\lambda'_{2\mathcal{T}}(t) > \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\},\ \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$.*

Proof. We consider the cases 1, 2.a and 3. In these cases, the size $\lambda'_{2\mathcal{T}}(t)$ of the constructed 2TPC is computed using Eqs. (7.3), (7.4) and (7.6) and the fact that $\lambda'_{2\mathcal{T}}(t) = |N_t| + |S_t| + |T_t|$. After performing simple computations, we conclude that in these cases the subroutine process(t) (cases 1, 2.a) returns $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$ if the following condition holds:

$$|N_\ell| - \max\{|s^1|, |s^2|\} > |F(V_r)| + |T_r|. \tag{7.7}$$

In case 3 subroutine process(t) returns $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|$ if the following condition holds:

$$|N_\ell| + \min\{s^1, s^2\} > |F(V_r)| + |T_r|. \tag{7.8}$$

We will show that (i) $\lambda'_{2\mathcal{T}}(t) > \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$ and, (ii) $\lambda'_{2\mathcal{T}}(t) > \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$. According to Proposition 7.1 and since $G[t]$ is a cograph, we have:

$$|\mathcal{T}_t^1| = |S_\ell^1| + |S_r^1| + |T_\ell| + |T_r| \text{ and } |\mathcal{T}_t^2| = |S_\ell^2| + |S_r^2| + |T_\ell| + |T_r|.$$

(i) We first show that $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)| > \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$. Consider the case where $s^1 \geq 0$ and $s^2 \geq 0$, and let $s^1 < s^2$; equivalently,

$$|S_\ell^1| - |S_r^2| < |S_\ell^2| - |S_r^1| \ \Leftrightarrow |S_\ell^1| + |S_r^1| + |T_\ell| + |T_r| < |S_\ell^2| + |S_r^2| + |T_\ell| + |T_r| \ \Leftrightarrow |\mathcal{T}_t^1| < |\mathcal{T}_t^2|.$$

By Proposition 7.1 and Eq. (7.7) we obtain $|N_\ell| + |S_\ell^1| + |S_\ell^2| + |T_\ell| - |F(V_r)| > |T_r| + \max\{s^1, s^2\} + |S_\ell^1| + |S_\ell^2| + |T_\ell| = |\mathcal{T}_t^2| - |S_r^2| + |S_t^1| + \max\{s^1, s^2\} = |\mathcal{T}_t^2| + s^1 + s^2 > |\mathcal{T}_t^2| = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$.

Since $\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)| = |N_\ell| + |S_\ell^1| + |S_\ell^2| + |T_\ell| - |F(V_r)|$, it follows that $\lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)| > \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$.

Now let $s^1 \geq s^2$; equivalently,

$$|S_\ell^1| - |S_r^2| \geq |S_\ell^2| - |S_r^1| \Leftrightarrow |\mathcal{T}_t^1| \geq |\mathcal{T}_t^2|.$$

By Proposition 7.1 and Eq. (7.7) we obtain $|N_\ell| + |S_\ell^1| + |S_\ell^2| + |T_\ell| - |F(V_r)| > |T_r| + \max\{s^1, s^2\} + |S_\ell^1| + |S_\ell^2| + |T_\ell| = |\mathcal{T}_t^1| - |S_r^1| + |S_\ell^2| + \max\{s^1, s^2\} = |\mathcal{T}_t^1| + s^2 + s^1 > |\mathcal{T}_t^1| = \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$.

Similarly, we can show that $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)| > \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}$ for the cases where $s^1 < 0$ and $s^2 < 0$, $s^1 \geq 0$ and $s^2 < 0$, and $s^1 < 0$ and $s^2 \geq 0$.

(ii) We next show that $\lambda'_{2\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$. Consider the case where $s^1 \geq 0$ and $s^2 \geq 0$. From Eq. (7.7) and since

$$|N_r| \leq |N_\ell| \leq |F(N_\ell)| \Leftrightarrow |N_r| \leq |F(V_\ell)| \Leftrightarrow |N_r| - |F(V_\ell)| \leq 0,$$

we obtain $|N_r| + |S_r^1| + |S_r^2| + |T_r| - |F(V_\ell)| < |N_r| + |S_r^1| + |S_r^2| - |F(V_\ell)| + |N_\ell| - \max\{s^1, s^2\} - |F(V_r)| < |S_r^1| + |S_r^2| + |N_\ell| - \max\{s^1, s^2\} - |F(V_r)| < |S_\ell^2| + |S_\ell^1| + |N_\ell| - |F(V_r)| \leq \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)|$.

Similarly, we can show that $\lambda'_{2\mathcal{T}}(t) = \lambda_{\mathcal{T}}(t_\ell) - |F(V_r)| > \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$ for the cases where $s^1 < 0$ and $s^2 < 0$, $s^1 \geq 0$ and $s^2 < 0$, and $s^1 < 0$ and $s^2 \geq 0$. ∎

Similarly we can show that if the subroutine process(t) returns a 2TPC of $G[t]$ of size $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|$ (case 2.b), then $\lambda'_{2\mathcal{T}}(t) > \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\},\ \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|\}$. Thus, we can prove the following result.

Lemma 7.5. *Let t be an S-node of $T(G)$ and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. The subroutine process(t) returns a 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of $G[t]$ of size*

$$\lambda'_{2\mathcal{T}}(t) = \begin{cases} \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1 & \textit{if } |N_\ell| = |V_r| \textit{ and} \\ & |S_\ell^1| = |S_\ell^2| = |\mathcal{T}_r^1| = |\mathcal{T}_r^2| = 0, \\ \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, & \\ \qquad \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\} & \textit{otherwise.} \end{cases}$$

Obviously, a minimum 2TPC of the graph $G[t]$ is of size $\lambda_{2\mathcal{T}}(t) \leq \lambda'_{2\mathcal{T}}(t)$. On the other hand, we have proved a lower bound for the size $\lambda_{2\mathcal{T}}(t)$ of a minimum 2TPC of the graph $G[t]$ (see Lemma 7.1), namely, $\lambda_{2\mathcal{T}}(t) \geq \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\},\ \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\}$. It follows that $\lambda'_{2\mathcal{T}}(t) = \lambda_{2\mathcal{T}}(t)$, and, thus, we can state the following result.

Lemma 7.6. *Subroutine process(t) returns a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of the graph $G[t]$, for every internal S-node $t \in T(G)$.*

Since the above result holds for every S-node t of the modified co-tree $T(G)$, it also holds when t is the root of $T(G)$ and $\mathcal{T}_t^1 = \mathcal{T}^1$ and $\mathcal{T}_t^2 = \mathcal{T}^2$. Thus, the following theorem holds:

Theorem 7.1. *Let G be a cograph and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. Let t be the root of the modified co-tree $T(G)$, and let $\mathcal{P}_{2\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{2\mathcal{T}}(t_r)$ be a minimum 2TPC of $G[t_\ell]$ and $G[t_r]$, respectively. Algorithm Minimum_2TPC correctly computes a minimum 2TPC of $G = G[t]$ with respect to $\mathcal{T}^1 = \mathcal{T}_t^1$ and $\mathcal{T}^2 = \mathcal{T}_t^2$ of size $\lambda_{2\mathcal{T}} = \lambda_{2\mathcal{T}}(t)$, where*

$$\lambda_{2\mathcal{T}}(t) = \begin{cases} \lambda_{2\mathcal{T}}(t_r) + \lambda_{2\mathcal{T}}(t_\ell) & \textit{if } t \textit{ is a P-node,} \\ \max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\} + 1 & \textit{if } t \textit{ is an S-node and} \\ & |N_\ell| = |V_r| \textit{ and} \\ & |S_\ell^1| = |S_\ell^2| = |\mathcal{T}_r^1| = |\mathcal{T}_r^2| = 0, \\ \max\{\max\{|\mathcal{T}_t^1|, |\mathcal{T}_t^2|\}, \lambda_{2\mathcal{T}}(t_\ell) - |F(V_r)|, & \\ \qquad \lambda_{2\mathcal{T}}(t_r) - |F(V_\ell)|\} & \textit{otherwise.} \end{cases}$$

Let G be a cograph on n vertices and m edges, $\mathcal{T}^1$ and $\mathcal{T}^2$ be two terminal sets, and let t be an S-node of the modified co-tree $T(G)$. From the description of the algorithm we can easily conclude that a minimum 2TPC $\mathcal{P}_{2\mathcal{T}}(t)$ of $G[t]$ can be constructed in $O(E(G[t]))$ time, since we use at most $|V(G[t_\ell])| \cdot |V(G[t_r])|$ edges to connect the paths of the minimum 2TPCs of the graphs $G[t_\ell]$ and $G[t_r]$; in the case where t is a P-node a minimum 2TPC is constructed in $O(1)$ time. Thus, the time needed by the subroutine process(t) to compute a minimum 2TPC in the case where t is the root of the tree $T(G)$ is $O(n + m)$; moreover, through the execution of the subroutine no additional space is needed. The construction of the co-tree $T_{co}(G)$ of G needs $O(n + m)$ time and it requires $O(n)$ space [22, 24]. Furthermore, the binarization process of the co-tree, that is, the construction of the modified co-tree $T(G)$, takes $O(n)$ time. Hence, we can state the following result.

Theorem 7.2. *Let G be a cograph on n vertices and m edges and let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two disjoint subsets of $V(G)$. A minimum 2-terminal-set path cover $\mathcal{P}_{2\mathcal{T}}$ of G can be computed in $O(n + m)$ time and space.*

7.5 Concluding Remarks

In this chapter, we introduce the 2-terminal-set path cover problem (2TPC) and propose a polynomial-time solution on the class of cographs. Our algorithm produces a minimum 2-terminal-set path cover of a cograph G that contains the largest number of terminal paths. It is worth investigating the existence of a linear-time algorithm for finding a minimum 2-terminal-set path cover on cographs that contains a large number of semi-terminal paths; we pose it as an open problem. It would also be interesting to see if the 2-terminal-set path cover problem can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. This promises to be an interesting area for further research since the complexity status of simpler problems, such as 1HP and 2HP, is unknown for interval graphs.

Chapter 8

Solving the $k(2)$-terminal-set Path Cover Problem on Cographs

8.1 Introduction

Another path cover related problem that has received increased attention in recent years is in the context of communication networks. The only efficient way to transmit high volume communication, such as in multimedia applications, is through disjoint paths that are dedicated to pairs of processors. To efficiently utilize the network one needs a simple algorithm that, with minimum overhead, constructs a large number of edge-disjoint paths between pairs of two given sets of requests. Motivated by this issue, we state a variant of the path cover problem, namely, the k-terminal-set path cover problem (kTSPC), which generalizes both 1HP and 2HP problems.

Problem kTSPC. Let G be a graph and let $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be disjoint sets of vertices of $V(G)$. A *k-terminal-set path cover* of the graph G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a path cover of G such that all vertices in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are endpoints of paths in the path cover and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ have one endpoint in $\mathcal{T}^i$ and the other in $\mathcal{T}^j$, $i \neq j$ and $i, j \in [1, k]$; a *minimum k-terminal-set path cover* of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a k-terminal-set path cover of G with minimum cardinality; the *k-terminal-set path cover problem* (kTSPC) is to find a minimum k-terminal-set path cover of the graph G.

Note that, if $\mathcal{T}^i$, $\forall i \in [1, k]$ is empty, the stated problem coincides with the classical path cover problem, while if $|\mathcal{T}^i| = 1$, $\forall i \in [1, k]$ the problem coincides with the k-fixed-endpoint path cover problem. In this chapter, we introduce a special case of the k(2)TSPC problem, namely, the $k(2)$-terminal-set path cover problem (k(2)TSPC). Since the complexity status of the k(2)TSPC problem on the class of cographs is unknown, we propose a polynomial-time solution for the k(2)TSPC problem on cographs. We next state the k(2)TSPC problem.

Problem k(2)TSPC. Let G be a graph and let $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be disjoint sets of vertices of $V(G)$ each containing at most two vertices. A *$k(2)$-terminal-set path cover* of the graph G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a path cover of G such that all vertices in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are endpoints of paths in the path cover and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ have one endpoint in $\mathcal{T}^i$ and the other in $\mathcal{T}^j$, $i \neq j$ and $i, j \in [1, k]$; a *minimum $k(2)$-terminal-set path cover* of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a $k(2)$-terminal-set path cover of G with minimum cardinality; the *$k(2)$-terminal-set path cover problem* (k(2)TSPC) is to find a minimum $k(2)$-terminal-set path cover of the graph G.

Contribution. We show that the $k(2)$-terminal-set path cover problem (k(2)TSPC) has a polynomial-time solution in the class of complement reducible graphs, or cographs [24]. More precisely, we establish a lower bound on the size of a minimum $k(2)$-terminal-set path cover of a cograph G on n vertices and m edges. We then define path operations, and prove structural properties for the paths of such a path cover, which enable us to describe a simple algorithm for the k(2)TSPC problem. The proposed algorithm runs in time linear in the size of the input graph G, that is, in $O(n + m)$ time, and requires linear space.

The proposed algorithm for the k(2)TSPC problem can also be used to solve the 1HP and 2HP problems on cographs within the same time and space complexity. Moreover, we have designed our algorithm so that it produces a minimum $k(2)$-terminal-set path cover of a cograph G that contains a large number of paths with one endpoint in $\mathcal{T}^i$, $i \in [1, k]$ and the other in $\mathcal{T}^j$, $j \in [1, k]$ and $i \neq j$ (we can easily find a graph G so that G admits two minimum $k(2)$-terminal-set path covers with different numbers of paths having one endpoint in $\mathcal{T}^i$ and the other in $\mathcal{T}^j$, $i \neq j$).

8.2 Theoretical Framework

We binarize the co-tree $T_{co}(G)$ of a cograph G in such a way that each of its internal nodes has exactly two children [61, 70]. We shall refer to the binarized version of $T_{co}(G)$ as the modified co-tree of G and will denote it by $T(G)$. Thus, the left and right child of an internal node t of $T(G)$ will be denoted by t_ℓ and t_r, respectively. Let t be an internal node of $T(G)$. Then $G[t]$ is the subgraph of G induced by the subset V_t of the vertex set $V(G)$, which contains all the vertices of G that have as common ancestor in $T(G)$ the node t. For simplicity, we will denote by V_ℓ and V_r the vertex sets $V(G[t_\ell])$ and $V(G[t_r])$, respectively.

Let G be a cograph, $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be disjoint sets of vertices of $V(G)$ such that each one contains at most one vertex, and let $\mathcal{P}_{k\mathcal{T}}(G)$ be a minimum $k(2)$-terminal-set path cover of G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of size $\lambda_{k\mathcal{T}}$; note that the size of $\mathcal{P}_{k\mathcal{T}}(G)$ is the number of paths it contains. The vertices of the sets $\mathcal{T}^i$, $1 \leq i \leq k$ are called *terminal* vertices, and the sets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ are called the *terminal sets* of G, while those of $V(G) - (\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k)$ are called *non-terminal or free* vertices. Furthermore, we say that a vertex u is a terminal vertex and denote it by $u \in \mathcal{T}$ if it belongs to any $\mathcal{T}^i$, $1 \leq i \leq k$. Thus, the set $\mathcal{P}_{k\mathcal{T}}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices belonging to different sets, that is, $u \in \mathcal{T}^i$ and $v \in \mathcal{T}^j$, $i \neq j$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}^i$, $1 \leq i \leq k$ and the other in $V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}^i$, $1 \leq i \leq k$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$; if P_f consists of only one vertex, say, u, then $u \in V(G) - (\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k)$.

The set of the non-terminal paths in a minimum k(2)TSPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Furthermore, we denote by S_i a set of semi-terminal paths having a terminal endpoint belonging to the terminal set $\mathcal{T}^i$. The following equation holds.

$$\lambda_{k\mathcal{T}} = |N| + |S| + |T| \tag{8.1}$$

From the definition of the $k(2)$-terminal-set path cover problem (k(2)TSPC), we can easily conclude that the number of paths in a minimum k(2)TSPC cannot be less than the number of the terminal vertices divided by two. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T|$. Thus, we have the following proposition, which also holds for general graphs:

Proposition 8.1. *Let G be a cograph and let $\mathcal{T}^1, \ldots, \mathcal{T}^k$ be disjoint subsets of $V(G)$. Then $\lambda_{k\mathcal{T}} \geq \lceil \frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2} \rceil$.*

Clearly, the size of a k(2)TSPC of a cograph G, as well as the size of a minimum k(2)TSPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_{k\mathcal{T}} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G; hereafter, $F(V) = F(V(G))$. Furthermore, let $\mathcal{P}$ be a set of paths and let $V_{\mathcal{P}}$ denote the set of vertices belonging to the paths of the set $\mathcal{P}$; hereafter, $F(\mathcal{P}) = F(V_{\mathcal{P}})$. Then, if $\mathcal{T}^1, \ldots, \mathcal{T}^k$ are disjoint subsets of $V(G)$, we have $\lambda_{k\mathcal{T}} \leq |F(V)| + |\mathcal{T}^1| + \ldots + |\mathcal{T}^k|$.

Let t be an internal node of the tree $T(G)$, that is, t is either an S-node or a P-node [68]. Then $\lambda_{k\mathcal{T}}(t)$ denotes the number of paths in a minimum k(2)TSPC of the graph $G[t]$ with respect to $\mathcal{T}_t^1, \ldots, \mathcal{T}_t^k$, where $\mathcal{T}_t^i$, $1 \leq i \leq k$ are the terminal vertices of $\mathcal{T}^i$ of the graph $G[t]$. Let t_ℓ and t_r be the left and the right child of node t, respectively. We denote by $\mathcal{T}_\ell^i$ and $\mathcal{T}_r^i$ the terminal vertices of $\mathcal{T}^i$ in V_ℓ and V_r, respectively, where $V_\ell = V(G[t_\ell])$ and $V_r = V(G[t_r])$. Furthermore, we denote by $\mathcal{T}_\ell$ and $\mathcal{T}_r$ the set containing the terminal vertices belonging in any $\mathcal{T}^i$ in V_ℓ and V_r, respectively. Let N_ℓ, S_ℓ and T_ℓ be the sets of the non-terminal, semi-terminal and terminal paths in a minimum k(2)TSPC of $G[t_\ell]$, respectively. Similarly, let N_r, S_r and T_r be the sets of the non-terminal, semi-terminal and terminal paths in a minimum k(2)TSPC of $G[t_r]$, respectively. Obviously, Eq. (8.1) holds for $G[t]$ as well, with t being either an S-node or a P-node, that is,

$$\lambda_{k\mathcal{T}}(t) = |N_t| + |S_t| + |T_t| \tag{8.2}$$

where N_t, S_t and T_t are the sets of the non-terminal, the semi-terminal and the terminal paths, respectively, in a minimum k(2)TSPC of $G[t]$, that is in $\mathcal{P}_{k\mathcal{T}}(t)$. If t is a P-node, then $\mathcal{P}_{k\mathcal{T}}(t) = \mathcal{P}_{k\mathcal{T}}(t_\ell) \cup \mathcal{P}_{k\mathcal{T}}(t_r)$, where $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ are minimum k(2)TSPCs corresponding to $G[t_\ell]$ and $G[t_r]$, respectively, and $\lambda_{k\mathcal{T}}(t) = \lambda_{k\mathcal{T}}(t_\ell) + \lambda_{k\mathcal{T}}(t_r)$. Furthermore, in the case where t is a P-node, we have

$$\begin{aligned} |N_t| &= |N_\ell| + |N_r| \\ |S_t| &= |S_\ell| + |S_r| \\ |T_t| &= |T_\ell| + |T_r| \end{aligned}$$

Thus, we focus on computing a minimum k(2)TSPC of the graph $G[t]$ for the case where t is an S-node. Before describing our algorithm, we establish a lower bound on the size $\lambda_{k\mathcal{T}}(t)$ of a minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of a graph $G[t]$. More precisely, we prove the following lemma.

Lemma 8.1. *Let t be an internal node of $T(G)$ and let $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ be a minimum k(2)TSPC of $G[t_\ell]$ and $G[t_r]$, respectively. Then $\lambda_{k\mathcal{T}}(t) \geq \max\{\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil + c,\ \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|\}$, where c is defined as follows:*

$$c = \begin{cases} 1 & \textit{if } |S_\ell| = 1 \textit{ and } |S_r| = 1 \textit{ and } |\mathcal{T}_\ell| = |\mathcal{T}_r| = 1 \textit{ and } k = 1, \\ 1 & \textit{if } |\mathcal{T}_\ell| = |S_\ell| = 2 \textit{ and } |\mathcal{T}_r| = 0 \textit{ (resp. } |\mathcal{T}_\ell| = 0 \textit{ and } |\mathcal{T}_r| = |S_r| = 2\textit{) and } k = 1, \\ 1 & \textit{if } |S_\ell| = |S^i_\ell| = 2 \textit{ and } |\mathcal{T}_r| = 0 \textit{ and } F(V_r) < 2 \textit{ and } |\mathcal{T}_\ell| > 0 \textit{ (resp. } |S_r| = |S^i_r| = 2 \textit{ and} \\ & |\mathcal{T}_\ell| = 0 \textit{ and } F(V_\ell) < 2 \textit{ and } |\mathcal{T}_r| > 0\textit{)} \\ 1 & \textit{if } |S_\ell| = 0 \textit{ and } |\mathcal{T}_r| = 0 \textit{ and } |N_\ell| = |V_r| \textit{ (resp. } |S_r| = 0 \textit{ and } |\mathcal{T}_\ell| = 0 \textit{ and } |N_r| = |V_\ell|\textit{)} \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Clearly, according to Proposition 2.1 and since $G[t]$ is a cograph, we have $\lambda_{k\mathcal{T}}(t) \geq \lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil$. Suppose that $|S_\ell| = 1$ and $|S_r| = 1$ and the two semi-terminal paths have terminal endpoints belonging to the same terminal set and $|\mathcal{T}_\ell| = |\mathcal{T}_r| = 1$. There cannot exist a minimum k(2)TSPC of $G[t]$ of size $\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil = 1$, since the two terminal vertices cannot belong to the same path. Suppose now that $|\mathcal{T}_\ell| = |S_\ell| = 2$ and $|\mathcal{T}_r| = 0$ and $k = 1$. Again, there cannot exist a minimum k(2)TSPC of $G[t]$ of size $\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil = 1$, since the two terminal vertices cannot belong to the same path. The same holds when $|\mathcal{T}_\ell| = 0$ and $|\mathcal{T}_r| = |S_r| = 2$ and $k = 1$. Now let $|S_\ell| = 2$ and $|\mathcal{T}_r| = 0$ and $F(V_r) < 2$ and $|\mathcal{T}_\ell| > 0$ and the two semi-terminal paths have terminal endpoints belonging to the same terminal set. Clearly, since the two semi-terminal paths have terminal endpoints belonging to the same terminal set, the size of a minimum k(2)TSPC of $G[t]$ cannot be less than $\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil + 1$. The same holds when $|\mathcal{T}_\ell| = 0$ and $|S_r| = 2$ and $F(V_\ell) < 2$ and $|\mathcal{T}_r| > 0$ and the two semi-terminal paths have terminal endpoints belonging to the same terminal set. Finally, it is easy to see that there cannot exist a minimum k(2)TSPC of size less than $\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil + 1$ when $|S_\ell| = 0$ (resp. $|S_r| = 0$) and $|\mathcal{T}_r| = 0$ (resp. $|\mathcal{T}_\ell| = 0$) and $|N_\ell| = |V_r|$ (resp. $|N_r| = |V_\ell|$). Indeed, we will show that $\lambda_{k\mathcal{T}}(t) \geq \lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil + 1 \Leftrightarrow \lambda_{k\mathcal{T}}(t) > \lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil$. Thus, we only need to prove that $\lambda_{k\mathcal{T}}(t) \neq \lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil$. Note that by the assumption we have $\lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil = |T_\ell|$. We assume that $\lambda_{k\mathcal{T}}(t) = \lceil\frac{|\mathcal{T}^1 \cup \ldots \cup \mathcal{T}^k|}{2}\rceil$, and, thus, $\lambda_{k\mathcal{T}}(t) = |T_\ell|$. There exists at least one non-terminal path in $G[t_\ell]$; for otherwise $|N_\ell| = 0$, and thus $V_r = \emptyset$, a contradiction. We ignore the terminal paths from the minimum k(2)TSPC of $G[t_\ell]$ and apply the algorithm described in [61] to $G[t]$. The resulting minimum k(2)TSPC contains only one (non-terminal) path which either has both endpoints in $G[t_\ell]$ or it has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. This non-terminal path can not be inserted into a terminal path of $G[t_\ell]$ because it does not have both endpoints in $G[t_r]$. Thus, $\lambda_{k\mathcal{T}}(t) = |T_\ell| + 1$, a contradiction.

We will prove that $\lambda_{k\mathcal{T}}(t) \geq \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$. Assume that $\lambda_{k\mathcal{T}}(t) < \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$. Consider removing from this path cover all the vertices in V_r. What results is a set of paths which is clearly a k(2)TSPC for $G[t_\ell]$. Since the removal of a free vertex in $F(V_r)$ will increase the number of paths by at most one, we obtain a k(2)TSPC of $G[t_\ell]$ of size at most $\lambda_{k\mathcal{T}}(t) + |F(V_r)|$. The assumption $\lambda_{k\mathcal{T}}(t) < \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|$ guarantees that $\lambda_{k\mathcal{T}}(t) + |F(V_r)| < \lambda_{k\mathcal{T}}(t_\ell)$, contradicting the minimality of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$. Using similar arguments we can show that $\lambda_{k\mathcal{T}}(t) \geq \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|$. Hence, the lemma follows. ∎

We next define four operations on paths of a minimum k(2)TSPC of the graphs $G[t_\ell]$ and $G[t_r]$, namely *break*, *connect*, *bridge* and *insert* operations; these operations are illustrated in Fig. 8.1.

- *Break* operation: Let $P = [p_1, p_2, \ldots, p_s]$ be a path of $\mathcal{P}_{k\mathcal{T}}(t_r)$ or $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ of length s. We say that we *break* the path P in two paths, say, P_1 and P_2, if we delete an arbitrary edge of P, say the edge

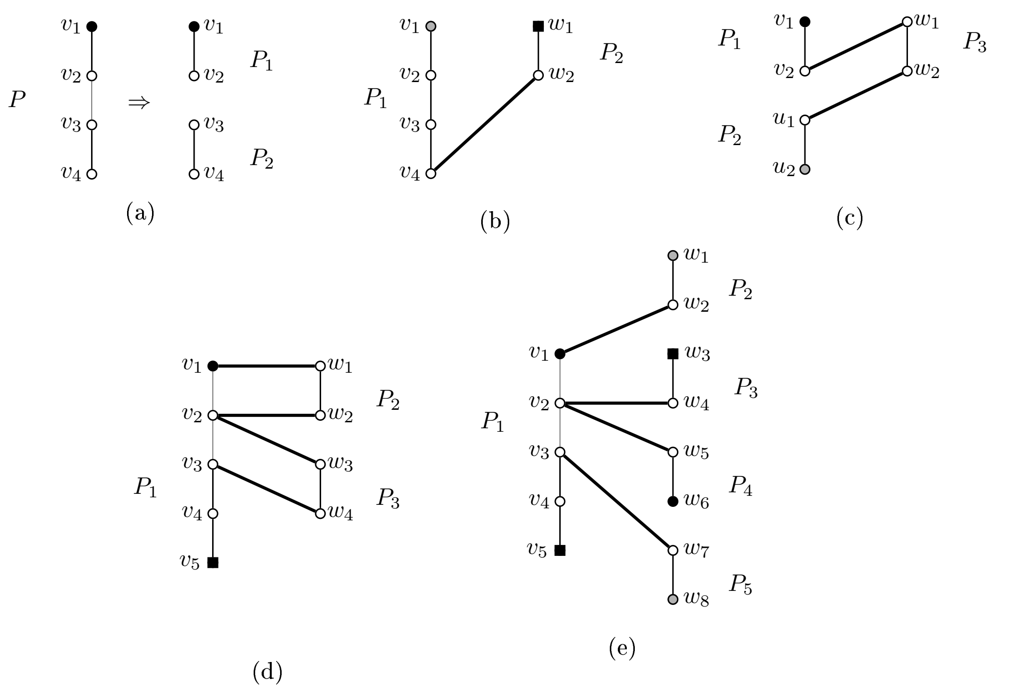


Figure 8.1: Illustrating (a) break, (b) connect, (c) bridge, (d) insert, and (e) connect-bridge operations; the vertices denoted by black-circles (resp. grey-circles) belong to $\mathcal{T}^i$ (resp. $\mathcal{T}^j$), while the vertices denoted by black-squares belong to $\mathcal{T}^p$, $i \neq j \neq p$.

$p_i p_{i+1}$ ($1 \leq i < s$), in order to obtain two paths which are $P_1 = [p_1, \ldots, p_i]$ and $P_2 = [p_{i+1}, \ldots, p_s]$. Note that we can break the path P in at most s trivial paths.

- *Connect* operation: Let P_1 be a non-terminal or a semi-terminal path of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_r)$) and let P_2 be a non-terminal or a semi-terminal path of $\mathcal{P}_{k\mathcal{T}}(t_r)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_\ell)$). We say that we *connect* the path P_1 with the path P_2, if we add an edge which joins two free endpoints of the two paths. Note that if $P_1 \in S_\ell$ (resp. $P_1 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^i$ then, if P_2 is also a semi-terminal path, $P_2 \in S_r$ (resp. $P_2 \in S_\ell$) with a terminal endpoint belonging to $\mathcal{T}^j$, $i \neq j$.

- *Bridge* operation: Let P_1 and P_2 be two paths of the set $N_\ell \cup S_\ell^1 \cup S_\ell^2$ (resp. $N_r \cup S_r^1 \cup S_r^2$) and let P_3 be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *bridge* the two paths P_1 and P_2 using path P_3 if we connect a free endpoint of P_1 with one endpoint of P_3 and a free endpoint of P_2 with the other endpoint of P_3. The result is a path having both endpoints in $G[t_\ell]$ (resp. $G[t_r]$). Note that if $P_1 \in S_\ell$ (resp. $P_1 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^i$ then, if P_2 is also a semi-terminal path, $P_2 \in S_\ell$ (resp. $P_2 \in S_r$) with a terminal endpoint belonging to $\mathcal{T}^j$, $i \neq j$.

- *Insert* operation: Let $P_1 = [t_1, p_1, \ldots, p_1', t_1']$ be a path in $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ (resp. $\mathcal{P}_{k\mathcal{T}}(t_r)$) and let $P_2 = [p_2, \ldots, p_2']$ be a non-terminal path of the set N_r (resp. N_ℓ). We say that we *insert* the path P_2 into P_1, if we replace the first edge of P_1, that is, the edge $t_1 p_1$, with the path $[t_1, p_2, \ldots, p_2', p_1]$. Thus, the resulting path is $P_1 = [t_1, p_2, \ldots, p_2', p_1, \ldots, p_1', t_1']$. Note that we can replace every edge of P_1 so that we can insert at most $|F'(\{P_1\})| + 1$ non-terminal paths, where $F'(\{P_1\})$ is the set of the free internal vertices belonging to the path P_1. If the path $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_1^r, \ldots, p_1', t_1']$ is constructed by connecting a path of $\mathcal{P}_{k\mathcal{T}}(t_\ell)$, say, $P_\ell = [t_1, p_1, \ldots, p_1^\ell]$ with a path of $\mathcal{P}_{k\mathcal{T}}(t_r)$, say, $P_r = [p_1^r, \ldots, p_1', t_1']$, then it obviously has one endpoint in $G[t_\ell]$ and the other in $G[t_r]$. In this case, if $P_2 \in N_\ell$ (resp. N_r) we can only replace the edges of P_1 that belong to $G[t_r]$ (resp. $G[t_\ell]$). On the other hand, if P_2 has one endpoint, say, p_2, in N_ℓ and the other, say, p_2', in N_r, we insert P_2 into P_1 as follows: $P_1 = [t_1, p_1, \ldots, p_1^\ell, p_2', \ldots, p_2, p_1^r, \ldots, p_1', t_1']$.

We can also combine the operations connect and bridge to perform a new operation which we call a *connect-bridge* operation; such an operation is depicted in Fig. 8.1(e) and is defined below.

- *Connect-Bridge* operation: Let $P_1 = [t_1, p_1, \ldots, p_z, t_1']$ be a terminal path of the set T_ℓ (resp. T_r), where $t_1 \in \mathcal{T}^i$ and $t_1' \in \mathcal{T}^j$, $i \neq j$, and let $P_2, P_3, \ldots, P_{\frac{s+1}{2}}$ be semi-terminal paths of the set S_r (resp. S_ℓ) with terminal endpoints belonging to pairwise disjoint sets $\mathcal{T}^p$ and $P_{\frac{s+1}{2}+1}, \ldots, P_s$ be semi-terminal paths of the set S_r (resp. S_ℓ) with terminal endpoints belonging to pairwise disjoint sets $\mathcal{T}^{p'}$, where s is odd and $3 \leq s \leq 2z+3$. Let P_2 have a terminal endpoint belonging to $\mathcal{T}^{i'}$, $i' \neq i$, and let P_s have a terminal endpoint belonging to $\mathcal{T}^{j'}$, $j' \neq j$. We say that we *connect-bridge* the paths $P_2, P_3, \ldots, P_s$ using vertices of P_1, if we perform the following operations: (i) connect the path P_2 with the path $[t_1]$; (ii) bridge $r = \frac{s-3}{2}$ pairs of different semi-terminal paths using vertices $p_1, p_2, \ldots, p_r$; and (iii) connect the path $[p_{r+1}, \ldots, p_z, t_1']$ with the last semi-terminal path P_s.

The Connect-Bridge operation produces two paths having one endpoint in $G[t_\ell]$ and the other endpoint in $G[t_r]$ and $\frac{s-3}{2}$ paths having both endpoints in $G[t_r]$ (resp. $G[t_\ell]$).

8.3 The Algorithm

We next present an optimal algorithm for the k(2)TSPC problem on cographs. Our algorithm takes as input a cograph G and k disjoint subsets $\mathcal{T}^1, \ldots, \mathcal{T}^k$ of its vertices, where $|\mathcal{T}^i| \leq 2$, $\forall i \in [1, k]$, and finds the paths of a minimum k(2)TSPC of G in linear time; it works as follows:

Algorithm Minimum_k(2)TSPC

Input: a cograph G and k disjoint subsets $\mathcal{T}^1, \ldots, \mathcal{T}^k$ of $V(G)$ such that $|\mathcal{T}^i| \leq 2$, $\forall i \in [1, k]$;

Output: a minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(G)$ of the cograph G;

1. Construct the co-tree $T_{co}(G)$ of G and make it binary; let $T(G)$ be the resulting tree;
2. Execute the subroutine *process*(*root*), where *root* is the root node of the tree $T(G)$; the minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(root) = \mathcal{P}_{k\mathcal{T}}(G)$ is the set of paths returned by the subroutine;

Algorithm 6: Algorithm Minimum_k(2)TSPC

where the description of the subroutine *process*( ) is as follows:

process (node t)
Input: node t of the modified co-tree $T(G)$ of the input graph G.
Output: a minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of the cograph $G[t]$.

```
1. if $t$ is a leaf
   then return($\{u\}$), where $u$ is the vertex associated with the leaf $t$;
   else {$t$ is an internal node that has a left and a right child denoted by $t_\ell$ and $t_r$, resp.}
        $\mathcal{P}_{k\mathcal{T}}(t_\ell) \leftarrow process(t_\ell)$;
        $\mathcal{P}_{k\mathcal{T}}(t_r) \leftarrow process(t_r)$;

2. if $t$ is a P-node
   then return($\mathcal{P}_{k\mathcal{T}}(t_\ell) \cup \mathcal{P}_{k\mathcal{T}}(t_r)$);

3. if $t$ is an S-node
   then $s = |S_\ell| - |S_r|$;
        case 1:  $s > 0$
                     call procedure $k(2)TSPC\_1$;
        case 2:  $s < 0$
                     call procedure $k(2)TSPC\_2$;
        case 3:  $s = 0$ and $\exists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$
                     call procedure $k(2)TSPC\_3$;
        case 4:  $s = 0$ and $\nexists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$
                     call procedure $k(2)TSPC\_4$;
```

We next describe the subroutine *process*( ) in the case where t is an S-node of $T(G)$. Thus, we distinguish the following four cases: (1) $s > 0$, (2) $s < 0$, (3) $s = 0$ and $\exists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$, and (4) $s = 0$ and $\nexists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$.

Case 1: $s > 0$

Let n_v be the number of vertices needed to bridge the semi-terminal paths of S_ℓ that cannot be connected to semi-terminal paths of S_r. Let SN_r be the set of non-terminal paths obtained by breaking the set $S_r \cup N_r$ into $\min\{|N_\ell| + n_v,\ |F(N_r \cup S_r)|\}$ non-terminal paths. We can construct the paths of a k(2)TSPC using the following procedure:

Procedure k(2)TSPC_1

1. break the paths of $N_r \cup S_r$ in order to obtain a set SN_r of $\min\{|N_\ell| + n_v,\ |F(N_r \cup S_r)|\}$ paths;
2. connect paths of S_ℓ with paths of S_r;
3. bridge paths of N_ℓ using at most $|SN_r| - n_v$ vertices of SN_r and if we obtain one path and $|SN_r| - (|N_\ell| - 1) > n_v$ connect its endpoints to paths of SN_r so that it has both endpoints in V_r. Let p be the resulting path;
4. bridge the rest of the paths in S_ℓ using paths from SN_r and also path p;
5. connect-bridge paths of S_ℓ using vertices of the paths belonging to T_r;
6. insert non-terminal paths of V_r into non-terminal paths of V_ℓ through vertices belonging to V_ℓ;
7. if $\exists$ a non-terminal path (created at Step 3) with one endpoint in V_r and the other in V_ℓ use a path created at Step 2 to insert it;
8. insert non-terminal paths of V_ℓ (resp. V_r) into paths of T_r (resp. $T_\ell \cup S_\ell$); (if $p \in T_r$ was used at Step 5 then only free vertices of p belonging to V_r are used)

We next describe Step 2 of the above procedure in detail. We first connect a semi-terminal path of the set S_ℓ^i, $1 \leq i \leq k$, with a semi-terminal path of the set S_r^j, $1 \leq j \leq k$ and $i \neq j$. We then mark the sets S^i and S^j as used sets. Next, we connect another semi-terminal path of S_ℓ^b, $1 \leq b \leq k$, with a semi-terminal path of S_r^c, $1 \leq c \leq k$ and $b \neq c$, where S^c is an unmarked set. Note that if there does not exist an unmarked set of semi-terminal paths, we use a marked set as long as $b \neq c$. The procedure continues until all semi-terminal paths of S_r are connected with paths of S_ℓ.

Case 2: $s < 0$

Let n_v be the number of vertices needed to bridge the semi-terminal paths of S_r that cannot be connected to semi-terminal paths of S_ℓ. Let SN_ℓ be the set of non-terminal paths obtained by breaking the set $S_\ell \cup N_\ell$ into $\min\{|N_r| + n_v,\ |F(N_\ell \cup S_\ell)|\}$ non-terminal paths. We can construct the paths of a k(2)TSPC using the following procedure:

Procedure k(2)TSPC_2

1. break the paths of $N_\ell \cup S_\ell$ in order to obtain a set SN_ℓ of $\min\{|N_r| + n_v,\ |F(N_\ell \cup S_\ell)|\}$ paths;
2. connect paths of S_r with paths of S_ℓ;
3. bridge paths of N_r using at most $|SN_\ell| - n_v$ vertices of SN_ℓ and if we obtain one path and $|SN_ell| - (|N_r| - 1) > n_v$ connect its endpoints to paths of SN_ℓ so that it has both endpoints in V_ℓ. Let p be the resulting path;
4. bridge the rest of the paths in S_r using paths from SN_ℓ and also path p;
5. connect-bridge paths of S_r using vertices of the paths belonging to T_ℓ;
6. insert non-terminal paths of V_ℓ into non-terminal paths of V_r through vertices belonging to V_r;
7. if $\exists$ a non-terminal path (created at Step 3) with one endpoint in V_r and the other in V_ℓ use a path created at Step 2 to insert it;
8. insert non-terminal paths of V_r (resp. V_ℓ) into paths of T_ℓ (resp. $T_r \cup S_r$); (if $p \in T_\ell$ was used at Step 5 then only free vertices of p belonging to V_ℓ are used)

We next describe Step 2 of the above procedure in detail. As described in Procedure k(2)TSPC_1, we first connect a semi-terminal path of the set S_ℓ^i, $1 \leq i \leq k$, with a semi-terminal path of the set S_r^j, $1 \leq j \leq k$ and $i \neq j$. We then mark the sets S^i and S^j as used sets. Next, we connect another semi-terminal path of S_ℓ^b, $1 \leq b \leq k$, with a semi-terminal path of S_r^c, $1 \leq c \leq k$ and $b \neq c$, where S^c is an unmarked set. Note that if there does not exist an unmarked set of semi-terminal paths, we use a marked set as long as $b \neq c$. The procedure continues until all semi-terminal paths of S_ℓ are connected with paths of S_r.

Case 3: $s = 0$ and $\exists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$

Let SN_r be the set of non-terminal paths obtained by breaking the set $S_r \cup N_r$ into $\min\{|N_\ell|,\ |F(N_r \cup S_r)|\}$ non-terminal paths. We can construct the paths of a k(2)TSPC using the following procedure:

Procedure k(2)TSPC_3

1. if $F(N_r) > F(N_\ell)$ swap$(\mathcal{P}_{k\mathcal{T}}(t_\ell), \mathcal{P}_{k\mathcal{T}}(t_r))$;
2. break the paths of $N_r \cup S_r$ in order to obtain a set SN_r of $\min\{|N_\ell|,\ |F(N_r \cup S_r)|\}$ paths;

3. connect paths of S_ℓ with paths of S_r;

4. bridge paths of N_ℓ using vertices of SN_r and if we obtain one path connect an endpoint to a path of SN_r so that it has one endpoint in V_r and the other in V_ℓ;

5. insert paths of SN_r into paths of N_ℓ through vertices belonging to V_ℓ;

6. if $\exists$ a non-terminal path (created at Step 4) with one endpoint in V_r and the other in V_ℓ use a path created at Step 2 to insert it;

7. insert paths of N_ℓ into paths of T_r;

Step 3 of the above procedure is similar to Step 2 of Procedures k(2)TSPC_1 and k(2)TSPC_2. The difference is that when Step 3 is finished all semi-terminal paths of S_r are connected with all semi-terminal paths of S_ℓ.

Case 4: $s = 0$ and $\nexists p \in S_\ell^i$ and $p' \in S_r^j$ such that $i \neq j$

Similarly to Case 3, let SN_r be the set of non-terminal paths obtained by breaking the set $S_r \cup N_r$ into $\min\{|N_\ell|,\ |F(N_r \cup S_r)|\}$ non-terminal paths. We can construct the paths of a k(2)TSPC using the following procedure:

Procedure k(2)TSPC_4

1. if $F(N_r) > F(N_\ell)$ swap$(\mathcal{P}_{k\mathcal{T}}(t_\ell), \mathcal{P}_{k\mathcal{T}}(t_r))$;

2. break the paths of $N_r \cup S_r$ in order to obtain a set SN_r of $\min\{|N_\ell|,\ |F(N_r \cup S_r)|\}$ paths;

3. bridge paths of N_ℓ using vertices of SN_r so that if we obtain one path and $|T_r| \neq 0$ (resp. $|T_\ell| \neq 0$) it has both its endpoints in V_ℓ (resp. V_r);

4. insert paths of SN_r into paths of N_ℓ through vertices belonging to V_ℓ;

5. insert paths of N_ℓ into paths of T_r;

6. if $|S_r| \neq 0$ connect a path of N_ℓ to the path of S_r;

8.4 Correctness and Time Complexity

Let G be a cograph, $T(G)$ be the modified co-tree of G, and let $\mathcal{T}^i$ $i \in [1, k]$ be the terminal sets of G. Since our algorithm computes a k(2)TSPC $\mathcal{P}'_{k\mathcal{T}}(t)$ of $G[t]$ of size $\lambda'_{k\mathcal{T}}(t)$ for each internal node $t \in T(G)$, and thus for the root $t = t_{root}$ of the tree $T(G)$, we need to prove that the constructed k(2)TSPC $\mathcal{P}'_{k\mathcal{T}}(t)$ is minimum. Obviously, the size $\lambda_{k\mathcal{T}}(t)$ of a minimum k(2)TSPC of the graph $G[t]$ is less than or equal to the size $\lambda'_{k\mathcal{T}}(t)$ of the k(2)TSPC constructed by our algorithm. After performing simple computations we get four specific values for the size $\lambda'_{k\mathcal{T}}(t)$ of the k(2)TSPC constructed by our algorithm, that is, by the k(2)TSPC procedures 1, 2, 3 and 4. More precisely, if t is an internal S-node of $T(G)$, our algorithm returns a k(2)TSPC of size $\lambda'_{k\mathcal{T}}(t)$ equal to $\max\{\lceil\frac{|\mathcal{T}^1\cup\ldots\cup\mathcal{T}^k|}{2}\rceil + c,\ \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|\}$, where c is defined as follows:

$$c = \begin{cases} 1 & \text{if } |S_\ell| = 1 \text{ and } |S_r| = 1 \text{ and } |\mathcal{T}_\ell| = |\mathcal{T}_r| = 1 \text{ and } k = 1, \\ 1 & \text{if } |\mathcal{T}_\ell| = |S_\ell| = 2 \text{ and } |\mathcal{T}_r| = 0 \text{ (resp. } |\mathcal{T}_\ell| = 0 \text{ and } |\mathcal{T}_r| = |S_r| = 2) \text{ and } k = 1, \\ 1 & \text{if } |S_\ell| = |S^i_\ell| = 2 \text{ and } |\mathcal{T}_r| = 0 \text{ and } F(V_r) < 2 \text{ and } |\mathcal{T}_\ell| > 0 \text{ (resp. } |S_r| = |S^i_r| = 2 \text{ and} \\ & |\mathcal{T}_\ell| = 0 \text{ and } F(V_\ell) < 2 \text{ and } |\mathcal{T}_r| > 0) \\ 1 & \text{if } |S_\ell| = 0 \text{ and } |\mathcal{T}_r| = 0 \text{ and } |N_\ell| = |V_r| \text{ (resp. } |S_r| = 0 \text{ and } |\mathcal{T}_\ell| = 0 \text{ and } |N_r| = |V_\ell|) \\ 0 & \text{otherwise.} \end{cases}$$

According to Lemma 8.1 the size $\lambda_{k\mathcal{T}}(t)$ of a minimum k(2)TSPC of the graph $G[t]$ is greater than or equal to the size $\lambda'_{k\mathcal{T}}(t)$ of the k(2)TSPC constructed by our algorithm. Thus, the size of a minimum k(2)TSPC of the graph $G[t]$ is equal to $\lambda'_{k\mathcal{T}}(t)$ and, consequently, the k(2)TSPC constructed by our algorithm is a minimum k(2)TSPC of $G[t]$. Thus, we can state the following result.

Lemma 8.2. *Subroutine process(t) returns a minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of the graph $G[t]$, for every internal S-node $t \in T(G)$.*

Since the above result holds for every S-node t of the modified co-tree $T(G)$, it also holds when t is the root of $T(G)$ and $\mathcal{T}^i_t = \mathcal{T}^i$, $i \in [1,k]$. Thus, the following theorem holds:

Theorem 8.1. *Let G be a cograph and let $\mathcal{T}^i$, $i \in [1,k]$ be k disjoint subsets of $V(G)$ such that $|\mathcal{T}^i| \le 2$. Let t be the root of the modified co-tree $T(G)$, and let $\mathcal{P}_{k\mathcal{T}}(t_\ell)$ and $\mathcal{P}_{k\mathcal{T}}(t_r)$ be a minimum k(2)TSPC of $G[t_\ell]$ and $G[t_r]$, respectively. Algorithm Minimum_k(2)TSPC correctly computes a minimum k(2)TSPC of $G = G[t]$ with respect to $\mathcal{T}^i = \mathcal{T}^i_t$, $i \in [1,k]$, of size $\lambda_{k\mathcal{T}} = \lambda_{k\mathcal{T}}(t)$, where*

$$\lambda_{k\mathcal{T}}(t) = \begin{cases} \lambda_{k\mathcal{T}}(t_r) + \lambda_{k\mathcal{T}}(t_\ell) & \textit{if } t \textit{ is a P-node,} \\ \\ \max\{\lceil\frac{|\mathcal{T}^1\cup\ldots\cup\mathcal{T}^k|}{2}\rceil + c,\ \lambda_{k\mathcal{T}}(t_\ell) - |F(V_r)|,\ \lambda_{k\mathcal{T}}(t_r) - |F(V_\ell)|\} & \textit{if } t \textit{ is an S-node} \end{cases}$$

and c is defined as follows:

$$c = \begin{cases} 1 & \textit{if } |S_\ell| = 1 \textit{ and } |S_r| = 1 \textit{ and } |\mathcal{T}_\ell| = |\mathcal{T}_r| = 1 \textit{ and } k = 1, \\ 1 & \textit{if } |\mathcal{T}_\ell| = |S_\ell| = 2 \textit{ and } |\mathcal{T}_r| = 0 \textit{ (resp. } |\mathcal{T}_\ell| = 0 \textit{ and } |\mathcal{T}_r| = |S_r| = 2\textit{) and } k = 1, \\ 1 & \textit{if } |S_\ell| = |S^i_\ell| = 2 \textit{ and } |\mathcal{T}_r| = 0 \textit{ and } F(V_r) < 2 \textit{ and } |\mathcal{T}_\ell| > 0 \textit{ (resp. } |S_r| = |S^i_r| = 2 \textit{ and} \\ & |\mathcal{T}_\ell| = 0 \textit{ and } F(V_\ell) < 2 \textit{ and } |\mathcal{T}_r| > 0\textit{)} \\ 1 & \textit{if } |S_\ell| = 0 \textit{ and } |\mathcal{T}_r| = 0 \textit{ and } |N_\ell| = |V_r| \textit{ (resp. } |S_r| = 0 \textit{ and } |\mathcal{T}_\ell| = 0 \textit{ and } |N_r| = |V_\ell|\textit{)} \\ 0 & \textit{otherwise.} \end{cases}$$

Let G be a cograph on n vertices and m edges, $\mathcal{T}^i$, $i \in [1,k]$ be k terminal sets, and let t be an S-node of the modified co-tree $T(G)$. From the description of the algorithm we can easily conclude that a minimum k(2)TSPC $\mathcal{P}_{k\mathcal{T}}(t)$ of $G[t]$ can be constructed in $O(E(G[t]))$ time, since we use at most $|V(G[t_\ell])|\cdot|V(G[t_r])|$ edges to connect the paths of the minimum k(2)TSPCs of the graphs $G[t_\ell]$ and $G[t_r]$; in the case where t is a P-node a minimum k(2)TSPC is constructed in $O(1)$ time. Thus, the time needed by the subroutine process(t) to compute a minimum k(2)TSPC in the case where t is the root of the tree $T(G)$ is $O(n+m)$; moreover, through the execution of the subroutine no additional space is needed. The construction of the co-tree $T_{co}(G)$ of G needs $O(n+m)$ time and it requires $O(n)$ space [22, 24]. Furthermore, the binarization process of the co-tree, that is, the construction of the modified co-tree $T(G)$, takes $O(n)$ time. Hence, we can state the following result.

Theorem 8.2. *Let G be a cograph on n vertices and m edges and let $\mathcal{T}^i$, $i \in [1,k]$, be k disjoint subsets of $V(G)$ such that $|\mathcal{T}^i| \leq 2$, $\forall i \in [1,k]$. A minimum $k(2)$-terminal-set path cover $\mathcal{P}_{k\mathcal{T}}$ of G can be computed in $O(n+m)$ time and space.*

8.5 Concluding Remarks

In this chapter, we introduce the $k(2)$-terminal-set path cover problem (k(2)TSPC) and propose a polynomial-time solution on the class of cographs. Our algorithm produces a minimum $k(2)$-terminal-set path cover of a cograph G that contains the largest number of terminal paths. The complexity status of the kTSPC problem on cographs is unknown, we pose it as an open problem. It would be interesting to see if the $k(2)$-terminal-set path cover problem can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. This promises to be an interesting area for further research since the complexity status of simpler problems, such as the 2HP problem, is unknown for interval graphs.

Chapter 9

A Polynomial Solution to the k-fixed-endpoint Path Cover Problem on Proper Interval Graphs

9.1 Introduction

The path cover problem is known to be NP-complete even when the input is restricted to several interesting special classes of graphs; for example, it is NP-complete on planar graphs [37], bipartite graphs [40], chordal graphs [40], chordal bipartite graphs [69] and strongly chordal graphs [69]. Bertossi and Bonuccelli [9] proved that the Hamiltonian Circuit problem is NP-complete on several interesting classes of intersection graphs.

Several variants of the HP problem are also of great interest, among which is the problem of deciding whether a graph admits a Hamiltonian path between two points (2HP). Recall that the 2HP problem is the same as the HP problem except that in 2HP two vertices of the input graph G are specified, say, u and v, and we are asked whether G contains a Hamiltonian path beginning with u and ending with v. Similarly, the 1HP problem is to determine whether a graph G admits a Hamiltonian path starting from a specific vertex u of G, and to find one if such a path does exist. Both 1HP and 2HP problems are also NP-complete in general graphs [36]. Some path cover related problems that have received both

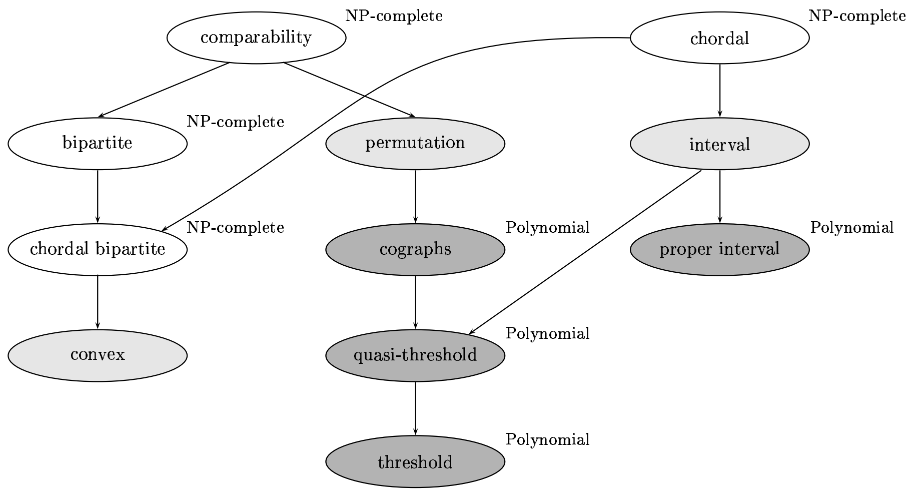


Figure 9.1: The complexity status (NP-complete, unknown, polynomial) of the kPC problem for some graph subclasses of comparability and chordal graphs. $A \rightarrow B$ indicates that class A contains class B.

theoretical and practical attention are in the content of communication and/or transposition networks [89] (see Section 5.1).

Motivated by the above issues we state a variant of the path cover problem, which is also described in Section 5.1, namely, the k-fixed-endpoint path cover problem (kPC), which generalizes both 1HP and 2HP, and also problem A described in Section 5.1.

(Problem kPC) Let G be a graph and let $\mathcal{T}$ be a set of k vertices of $V(G)$. A *k-fixed-endpoint path cover* of the graph G with respect to $\mathcal{T}$ is a path cover of G such that all vertices in $\mathcal{T}$ are endpoints of paths in the path cover; a *minimum k-fixed-endpoint path cover* of G with respect to $\mathcal{T}$ is a k-fixed-endpoint path cover of G with minimum cardinality; the *k-fixed-endpoint path cover problem* (kPC) is to find a minimum k-fixed-endpoint path cover of the graph G.

Contribution. In this chapter, we study the complexity status of the k-fixed-endpoint path cover problem (kPC) on the class of proper interval graphs, and show that this problem can be solved in polynomial time when the input is a proper interval graph [5, 13, 21, 29]. A *proper interval graph* is an interval graph that has an interval representation where no interval is properly contained in another. Proper interval graphs arise naturally in applications such as DNA sequencing [44]. The proposed algorithm runs in time linear in the size of the input graph G on n vertices and m edges, that is, in $O(n+m)$ time, and requires linear space. To the best of our knowledge, this is the first linear-time algorithm for solving the kPC problem on the class of proper interval graphs. The proposed algorithm for the kPC problem can also be used to solve the 1HP and 2HP problems on proper interval graphs within the same time and space complexity. Figure 9.1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status of the kPC problem, and thus, of 1HP and 2HP as well, on these classes; for definitions of the classes shown, see [13, 40]. Note that, if the problem is polynomially solvable on interval graphs, then it is also polynomially solvable on convex graphs [69].

Related Work. The class of proper interval graphs has been extensively studied in the literature [40, 80] and several linear-time algorithms are known for their recognition and realization [21, 29, 75]. Both Hamiltonian Circuit (HC) and Hamiltonian Path (HP) problems are polynomially solvable for the class of interval and proper interval graphs. Keil introduced a linear-time algorithm for the HC problem on interval graphs [58] and Arikati and Rangan [4] presented a linear-time algorithm for the minimum path cover problem on interval graphs. Bertossi [8] proved that a proper interval graph has a Hamiltonian path if and only if it is connected. He also gave an $O(n \log n)$ algorithm for finding a Hamiltonian circuit in a proper interval graph. Panda and Pas [75] presented a linear-time algorithm to detect if a proper interval graph is Hamiltonian. Furthermore, Lin et al. [61] proposed an optimal algorithm for the path cover problem on cographs while Nakano et al. [70] proposed an optimal parallel algorithm which finds and reports all the paths in a minimum path cover of a cograph in $O(\log n)$ time using $O(n/\log n)$ processors on a PRAM model. Hsieh et al. [49] presented an $O(n+m)$-time sequential algorithm for the Hamiltonian problem on a distance-hereditary graph and also proposed a parallel implementation of their algorithm which solves the problem in $O(\log n)$ time using $O((n+m)/\log n)$ processors on a PRAM model. A unified approach to solving the Hamiltonian problems on distance-hereditary graphs was presented in [50], while Hsieh [48] presented an efficient parallel strategy for the 2HP problem on the same class of graphs. Moreover, recently Nikolopoulos [72] solved the Hamiltonian problem on quasi-threshold graphs (a subclass of cographs) in $O(\log n)$ time using $O(n+m)$ processors on a PRAM model. Algorithms for the path cover problem on other classes of graphs were proposed in [51, 86].

Road Map. The chapter is organized as follows. In Section 9.2 we establish the notation and related terminology, and we present background results. In Section 9.3 we describe our linear-time algorithm for the kPC problem, while in Section 9.4 we prove its correctness and compute its time and space complexity. Finally, in Section 9.5 we conclude the chapter and discuss possible future extensions.

9.2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges. Let G be a graph; we denote its vertex set by $V(G)$ and its edge set by $E(G)$. Let $N(v) = \{w \in V(G) | vw \in E(G)\}$ be the set of neighbors of v and $N[v] = N(v) \cup \{v\}$. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$. If $G[N(v)]$ is a complete subgraph, then v is called a *simplicial vertex* of G.

9.2.1 Structural Properties of Proper Interval Graphs

A graph is a proper interval graph if and only if it is an interval graph with no induced subgraph isomorphic to the claw, which is the graph $K_{1,3}$ consisting of one vertex adjacent to three pairwise non-adjacent vertices [80]. The graph classes of proper interval graphs, interval graphs, and chordal graphs are *hereditary*: if G is in the class, then every induced subgraph of G is in the same class.

Let G be an interval graph and let $\{I_v = [a_v, b_v]\}$ be an interval representation of G; here, a_v and b_v ($a_v \leq b_v$) are referred to as the left and right endpoint of the interval I_v. The graph G is called a *unit* interval graph if all the intervals in the representation have unit length [64]. The family $\{I_V\}_{v \in V(G)}$ is the interval representation of a *proper* interval graph if no interval is properly contained in another. Clearly, unit interval graphs are proper interval graphs. Roberts [81] has proved the following fundamental result that shows that unit interval graphs, proper interval graphs, and indifference graphs are synonyms.

Proposition 9.1. *[81]: For a graph G, the following statements are equivalent:*
(i) G is a unit interval graph;
(ii) G is a proper interval graph;
(iii) G is an interval graph with no induced claw;
(iv) G is an indifference graph.

Proper interval graphs are characterized by an ordering of their vertices [64]:

Theorem 9.1. *[64]: A graph G is a proper interval graph if and only if there exists a linear order π on $V(G)$ such that for every choice of vertices u, v, w,*

$$u \prec_\pi v \prec_\pi w, \quad \textit{and} \quad uw \in E(G) \quad \textit{implies} \quad uv, vw \in E(G). \tag{9.1}$$

Let G be a proper interval graph on n vertices and m edges; an ordering π of the vertices of G satisfying (9.1) is referred to as *canonical* and it can be constructed in $O(n+m)$ time [64]. Theorem 9.1 implies the following result.

Corollary 9.1. *Let G be a proper interval graph and let π be a linear order on the vertex set of G satisfying (9.1). For every choice of subscripts i, j with $(1 \leq i < j \leq n)$ and $v_i v_j \in E(G)$, the vertices $v_i, v_{i+1}, \ldots, v_j$ are pairwise adjacent.*

9.2.2 Proper Interval Graphs and the kPC Problem

Let G be a proper interval graph with vertex set $V(G)$ and edge set $E(G)$, $\mathcal{T}$ be a set of k vertices of $V(G)$, and let $\mathcal{P}_\mathcal{T}(G)$ be a minimum k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ of size $\lambda_\mathcal{T}$; note that the size of $\mathcal{P}_\mathcal{T}(G)$ is the number of paths it contains. The vertices of the set $\mathcal{T}$ are called *terminal* vertices, and the set $\mathcal{T}$ is called the *terminal set* of G, while those of $V(G) - \mathcal{T}$ are called *non-terminal or free* vertices. Thus, the set $\mathcal{P}_\mathcal{T}(G)$ contains three types of paths, which we call *terminal*, *semi-terminal*, and *non-terminal or free* paths:

(i) a *terminal path* P_t consists of at least two vertices and both its endpoints, say, u and v, are terminal vertices, that is, $u, v \in \mathcal{T}$;

(ii) a *semi-terminal path* P_s is a path having one endpoint in $\mathcal{T}$ and the other in $V(G) - \mathcal{T}$; if P_s consists of only one vertex (trivial path), say, u, then $u \in \mathcal{T}$;

(iii) a *non-terminal or free path* P_f is a path having both its endpoints in $V(G) - \mathcal{T}$; if P_f consists of only one vertex, say, u, then $u \in V(G) - \mathcal{T}$.

Note that all the internal vertices of the paths of $\mathcal{P}_\mathcal{T}(G)$ are free vertices. Moreover, a semi-terminal path may consist of only one vertex which is a terminal vertex, while a terminal path contains at least two vertices. The set of the non-terminal paths in a minimum kPC of the graph G is denoted by N, while S and T denote the sets of the semi-terminal and terminal paths, respectively. Thus, we have

$$\lambda_\mathcal{T} = |N| + |S| + |T|. \tag{9.2}$$

From the definition of the k-fixed-endpoint path cover problem (kPC), we can easily conclude that the number of paths in a minimum kPC cannot be less than the number of terminal vertices divided by two. Furthermore, since each semi-terminal path contains one terminal vertex and each terminal path contains two, the number of terminal vertices is equal to $|S| + 2|T|$. Thus, the following proposition holds:

Proposition 9.2. *Let G be a proper interval graph and let $\mathcal{T}$ be a terminal set of G. Then $|\mathcal{T}| = |S| + 2|T|$ and $\lambda_\mathcal{T} \geq \lceil \frac{|\mathcal{T}|}{2} \rceil$.*

Clearly, the size of a kPC of a proper interval graph G, as well as the size of a minimum kPC of G, is less than or equal to the number of vertices of G, that is, $\lambda_\mathcal{T} \leq |V(G)|$. Let $F(V(G))$ be the set of the free vertices of G. Then we have the following proposition:

Proposition 9.3. *Let G be a proper interval graph and let $\mathcal{T}$ be a terminal set of G. If $\lambda_\mathcal{T}$ is the size of a minimum kPC of G, then $\lambda_\mathcal{T} \leq |F(V(G))| + |\mathcal{T}|$.*

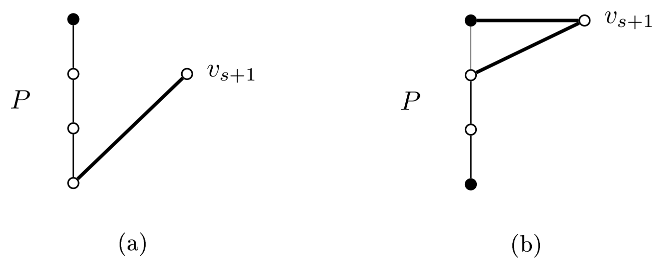


Figure 9.2: Illustrating (a) connect and (b) insert operations; $P \in \mathcal{P}_{\mathcal{T}}(G[\{v_1, v_2, \dots, v_s\}])$.

Our algorithm is based on a greedy principle, visiting the intervals according to the canonical ordering. From now on, by v_i, $1 \leq i \leq n$, we mean the vertex of G that is numbered with integer i. We say that $v_i < v_j$ if $i < j$, $1 \leq i, j \leq n$. Let $G[S]$ be the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \dots, v_s\}$, $1 \leq s < n$ and let $\mathcal{P}_{\mathcal{T}}(G[S])$ be a minimum kPC of $G[S]$. Before describing our algorithm, we define two operations on paths of a minimum kPC $\mathcal{P}_{\mathcal{T}}(G[S])$ of a proper interval graph $G[S]$ and a vertex $v_{s+1} \in V(G)$, namely *connect* and *insert* operations; these operations are illustrated in Fig. 9.2.

- *Connect* operation: Let P be a non-terminal or a semi-terminal path of $\mathcal{P}_{\mathcal{T}}(G[S])$, where $G[S]$ is the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \dots, v_s\}$, $1 \leq s < n$, and let v_{s+1} be a free or a terminal vertex. We say that we *connect* vertex v_{s+1} to the path P, if we add an edge which joins vertex v_{s+1} with a free endpoint of P.

- *Insert* operation: Let $P = [t, p, \dots, p', t']$ be a terminal or a semi-terminal path of $\mathcal{P}_{\mathcal{T}}(G[S])$, where $G[S]$ is the induced subgraph of G consisting of vertices $S = \{v_1, v_2, \dots, v_s\}$, $1 \leq s < n$, and let v_{s+1} be a free vertex. We say that we *insert* vertex v_{s+1} into P, if we replace an edge of P, say, the edge tp, with the path $[t, v_{s+1}, p]$. Thus, the resulting path is $P = [t, v_{s+1}, p, \dots, p', t']$.

Let P be a path of $\mathcal{P}_{\mathcal{T}}(G)$ and let v_i and v_j be its endpoints. We say that v_i is the left (resp. right) endpoint of the path and v_j is the right (resp. left) endpoint of the path if $v_i < v_j$ (resp. $v_j < v_i$). Furthermore, we say that v_i is an *available endpoint* of P if (i) $v_i \in V(G) - \mathcal{T}$, that is, v_i is a free endpoint, or (ii) $v_i \in \mathcal{T}$ and $P = [v_i]$.

9.3 The Algorithm

We next present an algorithm for the kPC problem on proper interval graphs. Our algorithm takes as input a proper interval graph G and a subset $\mathcal{T}$ of its vertices and finds the paths of a minimum kPC of G in linear time; it is based on the following greedy principle to extend a path $P = [p_1, p_2, \dots, p_s]$, $s \geq 1$, in a proper interval graph. Specifically, we start from a single vertex path $P_1 = [v_1]$ and continue by visiting the next vertex, that is, v_2. If v_2 is adjacent to v_1, we extend path P_1 to v_2, that is, we connect v_2 to the path P_1. Hence, path P_1 now consists of two vertices, that is, $P_1 = [v_1, v_2]$, and v_1 and v_2 are its left and right endpoints, respectively. It is worth pointing out that if a vertex is adjacent to both endpoints of a non-terminal path, we connect it to the left endpoint of the path. Suppose now that $v_1, v_2 \in \mathcal{T}$ and $N(v_3) = \{v_2\}$. Then, we start a new single vertex path $P_2 = [v_3]$. On the other hand, if $v_1, v_2 \in \mathcal{T}$ and $N(v_3) = \{v_1, v_2\}$, we insert v_3 into P_1; the resulting path is $P_1 = [v_1, v_3, v_2]$. Specifically, the algorithm works as follows:

Algorithm Minimum_kPC

Input: a proper interval graph G on n vertices and m edges and a set $\mathcal{T} \subseteq V(G)$;

Output: a minimum kPC $\mathcal{P}_\mathcal{T}(G)$ of the proper interval G;

1. Construct the canonical ordering π of the vertices of G;
2. Execute the subroutine *process*(π); the minimum kPC $\mathcal{P}_\mathcal{T}(G)$ is the set of paths returned by the subroutine;

Algorithm 7: Algorithm Minimum_kPC_ProperInterval

where the description of the subroutine *process*(π) is presented at Fig. 9.3.

Let P be a path in the kPC $\mathcal{P}_\mathcal{T}(G)$ constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively. Then, there is no path $P' \in \mathcal{P}_\mathcal{T}(G)$ having an endpoint between vertices v_i and v_j. Hence, we prove the following lemma:

Lemma 9.1. *Let P_s, $1 \leq s \leq \lambda_\mathcal{T}$ be a path in the kPC $\mathcal{P}_\mathcal{T}(G)$ of G constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively. Then, there is no path $P_t \in \mathcal{P}_\mathcal{T}(G)$, $1 \leq t \leq \lambda_\mathcal{T}$, $t \neq s$, such that $v_i < v_k < v_j$ or $v_i < v_\ell < v_j$, where v_k and v_ℓ are the left and right endpoints of P_t, respectively.*

Proof. Let P_s, $1 \leq s \leq \lambda_\mathcal{T}$ be a path in the kPC $\mathcal{P}_\mathcal{T}(G)$ constructed by Algorithm Minimum_kPC and let v_i and v_j be its left and right endpoints, respectively. Let $P_t \in \mathcal{P}_\mathcal{T}(G)$, $1 \leq t \leq \lambda_\mathcal{T}$, $t \neq s$, and let v_k and v_ℓ be its left and right endpoints, respectively. Suppose that $v_i < v_k < v_j$. Since v_i and v_j are the endpoints of P_s and $v_i < v_k < v_j$, the path P_s contains at least one edge, say, $v_a v_b$, such that $v_a < v_k < v_b$. Clearly, vertices v_a and v_b are free vertices. Since $v_a v_b \in E(G)$, we also have $v_a v_k \in E(G)$ and $v_k v_b \in E(G)$. Then, according to Algorithm Minimum_kPC, when vertex v_b is processed, it is either connected to the subpath of P_s which is already constructed having v_a as an endpoint or it is included to P_s through an edge $v_{a'} v_a$. If v_a is an endpoint when v_b is processed, it was also an endpoint when v_k was processed, and thus the edge $v_a v_k$ would appear to the path P_s, a contradiction. On the other hand, if v_b is included to P_s through the edge $v_{a'} v_a$, we have $v_{a'} < v_b$, and, thus, $v_{a'} v_k \in E(G)$; then v_k would appear to the path P_s, a contradiction. Similarly, we can prove that $v_\ell < v_i$ or $v_\ell > v_j$. ∎

Similarly, we can prove that, if P is a path in the kPC $\mathcal{P}_\mathcal{T}(G)$ constructed by Algorithm Minimum_kPC with v_i and v_j being its leftmost and rightmost vertices, respectively, there is no vertex between vertices v_i and v_j belonging to another path $P' \in \mathcal{P}_\mathcal{T}(G)$. Hence, we have the following lemma:

Lemma 9.2. *Let P_s, $1 \leq s \leq \lambda_\mathcal{T}$ be a path in the kPC $\mathcal{P}_\mathcal{T}(G)$ of G constructed by Algorithm Minimum_kPC and let v_i and v_j be its leftmost and rightmost vertices, respectively. Then, there is no vertex v_k belonging to another path $P_t \in \mathcal{P}_\mathcal{T}(G)$, $1 \leq t \leq \lambda_\mathcal{T}$, $t \neq s$, such that $v_i < v_k < v_j$.*

If $S = \{v_1, v_2, \ldots, v_{i-1}\}$, then $\mathcal{P}_\mathcal{T}(G[S])$ is the kPC of $G[S]$ constructed by the algorithm after processing $i-1$ vertices; let s be its size, where $1 \leq s \leq \lambda_\mathcal{T}$. It is easy to see that, if vertex v_i can be connected to the path P_s having the rightmost endpoints, then it is the only path it can be connected to. Thus, we prove the following lemma:

```
process ($\pi$)
Input: the canonical ordering of the vertices of $G$.
Output: a minimum kPC $\mathcal{P}_\mathcal{T}(G)$ of $G$.
$P_1 = [v_1]$; $i = 2$; $\lambda_\mathcal{T} = 1$;
$p_\ell = v_1$;      {the left endpoint of the path $P_{\lambda_\mathcal{T}}$}
$p_r = v_1$;      {the right endpoint of the path $P_{\lambda_\mathcal{T}}$}
while $i \le n$ do
    if $v_i p_\ell \in E(G)$ then
        case 1:  if $p_\ell = p_r$ or $p_\ell \notin \mathcal{T}$ then
                     connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through vertex $p_\ell$;
                     $p_\ell = p_r$; $p_r = v_i$;
        case 2:  if $p_\ell \in \mathcal{T}$ and $p_r \notin \mathcal{T}$ then
                     connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through vertex $p_r$;
                     $p_r = v_i$;
    else-if $v_i p_r \in E(G)$ and $p_r \notin \mathcal{T}$ then
        connect $v_i$ to $P_{\lambda_\mathcal{T}}$ through $p_r$;
        $p_r = v_i$;
    else-if $v_s, v_t$ ($s < i$, $t < i$) are the rightmost successive
            neighbors of $v_i$ in $P_{\lambda_\mathcal{T}}$ then
        insert $v_i$ into $P_{\lambda_\mathcal{T}}$ through the edge $p_s p_t$;
    else
        $\lambda_\mathcal{T} = \lambda_\mathcal{T} + 1$;
        $P_{\lambda_\mathcal{T}} = v_i$;
        $p_\ell = v_i$; $p_r = v_i$;
    $i = i + 1$;
endwhile;
$\mathcal{P}_\mathcal{T}(G) = \{P_1, \ldots, P_{\lambda_\mathcal{T}}\}$.
```

Figure 9.3: The subroutine *process*(π) of the Algorithm Minimum_kPC.

Lemma 9.3. *Let $\mathcal{P}_\mathcal{T}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$; let s be its size, where $1 \le s \le \lambda_\mathcal{T}$. If vertex v_i can be connected to the path P_s having the rightmost endpoints, then P_s is the only path that v_i can be connected to.*

Proof. Let $\mathcal{P}_\mathcal{T}(G[S])$ be the kPC of the graph $G[S]$ constructed by Algorithm Minimum_kPC after processing $i-1$ vertices, and let P_s be the path having the rightmost endpoints, say v_k and v_ℓ, $k < \ell$. Let P_{s-1} be the path with the rightmost endpoints in $\mathcal{P}_\mathcal{T}(G[S]) - \{P_s\}$, say v_r and v_t, $r < t$. Then, according to Lemma 9.1, $v_r < v_t < v_k < v_\ell$ and vertex v_i sees v_k and/or v_ℓ. Suppose that v_i can also be connected to the path P_{s-1}. If $v_i v_t \in E(G)$ and $v_t \notin \mathcal{T}$, then $v_k v_t \in E(G)$ and v_k is connected to the path P_{s-1}, a contradiction. If $v_t \in \mathcal{T}$ and $v_i v_r \in E(G)$ and $v_r \notin \mathcal{T}$, then $v_k v_r \in E(G)$ and v_k is connected to the path P_{s-1}, a contradiction. Thus, the lemma follows. ∎

Furthermore, as it is easy to see by induction, when vertex v_i is processed according to the algorithm, unless vertices $v_{i-2}, v_{i-1} \in \mathcal{T}$, they are successive in a path in $\mathcal{P}_\mathcal{T}(G[S])$ and/or at least one of them is an available endpoint. Hence, we prove the following lemma.

Lemma 9.4. *Let $\mathcal{P}_\mathcal{T}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$; let s be its size, where $1 \le s \le \lambda_\mathcal{T}$. Unless vertices $v_{i-2}, v_{i-1} \in \mathcal{T}$, they are successive in a path in $\mathcal{P}_\mathcal{T}(G[S])$ and/or at least one of them is an available endpoint.*

Proof. We use induction on i, $i \geq 3$. The basis $i = 3$ is trivial. Suppose that the hypothesis holds for $i-1$: let $\mathcal{P}_{\mathcal{T}}(G[S'])$ be the kPC of $G[S']$ constructed by Algorithm Minimum_kPC, where $S' = \{v_1, v_2, \ldots, v_{i-2}\}$ and let s' be its size, where $1 \leq s' \leq \lambda_{\mathcal{T}}$; then, unless vertices $v_{i-3}, v_{i-2} \in \mathcal{T}$, they are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S'])$ and/or at least one of them is an available endpoint. We show that, if $\mathcal{P}_{\mathcal{T}}(G[S])$ is the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$, and s is its size, where $1 \leq s \leq \lambda_{\mathcal{T}}$, then, unless vertices $v_{i-2}, v_{i-1} \in \mathcal{T}$, they are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$ and/or at least one of them is an available endpoint.

Clearly, $v_{i-2}v_{i-1} \in E(G)$, otherwise $G[S]$ is disconnected. We distinguish the following cases:

Case 1: When vertex v_{i-2} was processed, it was connected to a path or it constructed a new one; let P be the path containing v_{i-2}. Then, vertex v_{i-1} is either connected to the path P or it is inserted to P, or a new path is constructed containing only vertex v_{i-1}. If v_{i-1} is connected to the path P or a new path is constructed containing only v_{i-1}, then v_{i-2}, v_{i-1} are successive in P and/or v_{i-1} is an available endpoint. Consider the case where v_{i-1} is inserted into P, that is, v_{i-2} is not an available endpoint; then, unless $v_{i-3}, v_{i-2} \in \mathcal{T}$, by the induction hypothesis, v_{i-3} and v_{i-2} are successive into P and/or v_{i-3} is an available endpoint. If $v_{i-3}, v_{i-2} \in \mathcal{T}$, they are either successive in P, or, according to Lemma 9.2, v_{i-2}, v_{i-4} are successive in P. Thus, if v_{i-1} is inserted into P, then v_{i-2} and v_{i-1} are successive. Consequently, vertices v_{i-2}, v_{i-1} are successive in the path P and/or at least one of them is an available endpoint.

Case 2: When vertex v_{i-2} was processed, it was inserted into a path, say, P. Consequently, $v_{i-2} \in V(G) - \mathcal{T}$ and it has at least two neighbors. Vertex v_{i-1} does not see any endpoint since, if it did, v_{i-2} would also see it. Thus, v_{i-1} either constructs a new path or it is inserted into the path P. By the induction hypothesis, v_{i-3} and v_{i-2} are successive into P or v_{i-3} is an available endpoint. Thus, vertices v_{i-2}, v_{i-1} are successive in the path P and/or at least one of them is an available endpoint. Thus, the lemma follows. ∎

Generalizing the above lemma, we obtain the following:

Lemma 9.5. *Let $\mathcal{P}_{\mathcal{T}}(G[S])$ be the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$; let s be its size, where $1 \leq s \leq \lambda_{\mathcal{T}}$, and let $|N(v_i)| > 2$. There exists a neighbor of v_i being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$ and/or there exist two vertices $v_a, v_b \in N(v_i)$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$.*

Proof. We use induction on i, $i \geq 4$. The basis $i = 4$ is trivial. Suppose that the hypothesis holds for $i-1$: let $\mathcal{P}_{\mathcal{T}}(G[S'])$ be the kPC of $G[S']$ constructed by Algorithm Minimum_kPC, where $S' = \{v_1, v_2, \ldots, v_{i-2}\}$, let s' be its size, where $1 \leq s' \leq \lambda_{\mathcal{T}}$, and $|N(v_{i-1})| > 2$; then, there exists a neighbor of v_{i-1} being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S'])$ and/or there exist two vertices $v_{a'}, v_{b'} \in N(v_{i-1})$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S'])$. We show that, if $\mathcal{P}_{\mathcal{T}}(G[S])$ is the kPC of $G[S]$ constructed by Algorithm Minimum_kPC, where $S = \{v_1, v_2, \ldots, v_{i-1}\}$, s is its size, where $1 \leq s \leq \lambda_{\mathcal{T}}$, and $|N(v_i)| > 2$, then, there exists a neighbor of v_i being an available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$ and/or there exist two vertices $v_a, v_b \in N(v_i)$ that are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$.

If v_{i-1} and v_{i-2} are not both terminal vertices, according to Lemma 9.4, the lemma holds. If $v_{i-1}, v_{i-2} \in \mathcal{T}$, we distinguish the following cases:

Case 1: Vertices v_{i-1}, v_{i-2} belong to different paths. According to Lemma 9.1, vertex v_{i-2} belongs to a path consisting of only one vertex; thus, the lemma holds.

Case 2: Vertices v_{i-1}, v_{i-2} belong to a path consisting of only two vertices; then, the lemma holds.

Case 3: Vertices v_{i-1}, v_{i-2} belong to a path P consisting of more than two vertices; then, vertex v_{i-3} belongs to P (see Lemma 9.2), that is, $v_{i-3} \in V(G) - \mathcal{T}$, and, according to Lemma 9.4, when v_{i-1} is processed, v_{i-3} is an available endpoint or v_{i-3}, v_{i-2} are successive in P. Consequently, the lemma follows. ∎

The above lemma also shows that, when vertex v_i is inserted into a path P, it is inserted through the edge $v_{i-2}v_{i-1}$ or $v_{i-3}v_{i-1}$.

9.4 Correctness and Time Complexity

Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. In order to prove the correctness of Algorithm Minimum_kPC, we use induction on n. We also prove a property of the minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of G constructed by our algorithm: if $v_i \in N[v_n]$, $1 < i \leq n$ is the rightmost available endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$, then there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ having $v_j \in N[v_n]$, $1 < j \leq n$ as the rightmost available endpoint such that $v_i < v_j$. Hence, we prove the following theorem.

Theorem 9.2. *Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. Algorithm Minimum_kPC computes a minimum k-fixed-endpoint path cover $\mathcal{P}_{\mathcal{T}}(G)$ of G, having $v_i \in N[v_n]$, $1 < i \leq n$ as the rightmost available endpoint of its paths, such that there is no other minimum k-fixed-endpoint path cover $\mathcal{P}'_{\mathcal{T}}(G)$ having a rightmost available endpoint $v_j \in N[v_n]$, $1 < j \leq n$ such that $v_i < v_j$.*

Proof. We use induction on n. The basis $n = 1$ is trivial. Assume that Algorithm Minimum_kPC computes a minimum kPC $\mathcal{P}_{\mathcal{T}}(G[S])$ of every proper interval graph $G[S]$, $S = \{v_1, v_2, \ldots, v_{n-1}\}$, with at most $n - 1$ vertices having $v_i \in N[v_{n-1}]$, $1 < i \leq n - 1$, as the rightmost available endpoint of its paths, such that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G[S])$ having a rightmost available endpoint $v_j \in N[v_{n-1}]$, $1 < j \leq n - 1$, such that $v_i < v_j$; let $\lambda_{\mathcal{T}}(G[S])$ be the size of $\mathcal{P}_{\mathcal{T}}(G[S])$. We show that the algorithm computes a minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of every proper interval graph G with n vertices having $v_k \in N[v_n]$, $i \leq k \leq n$, as the rightmost available endpoint of its paths, such that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ having a rightmost available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$, such that $v_k < v_\ell$; let $\lambda_{\mathcal{T}}(G)$ be the size of $\mathcal{P}_{\mathcal{T}}(G)$.

Clearly, the size $\lambda'_{\mathcal{T}}(G)$ of a minimum kPC of G is $\lambda_{\mathcal{T}}(G)$ or $\lambda_{\mathcal{T}}(G) + 1$. Suppose that, when the algorithm processes vertex v_n, it connects it to an existing path, and, thus, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$; consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum kPC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$. Furthermore, if $v_n \notin \mathcal{T}$ then $\mathcal{P}_{\mathcal{T}}(G)$ contains a path having vertex v_n as an available endpoint, which is clearly the rightmost available endpoint in $N[v_n]$ that any other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ can contain. Consider the case where $v_n \in \mathcal{T}$. Suppose that the rightmost available endpoint in $N[v_n]$ of $\mathcal{P}_{\mathcal{T}}(G)$ is the same as the rightmost available endpoint in $N[v_n]$ of $\mathcal{P}_{\mathcal{T}}(G[S])$, that is, vertex v_i. Let $\mathcal{P}'_{\mathcal{T}}(G)$ be a kPC of G having a rightmost available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$, such that $v_i < v_\ell$. Clearly, $v_\ell \neq v_n$. Removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum kPC of $G[S]$ having $v_\ell \in N[v_n]$ as an available endpoint; a contradiction since v_i is the rightmost endpoint of any minimum kPC of $G[S]$. Now suppose that vertex v_n is connected to vertex v_i. Then, according to the algorithm, there exists no available endpoint in $N[v_n]$ in $\mathcal{P}_{\mathcal{T}}(G)$. We show that there is no other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ containing an available endpoint in $N[v_n]$. Indeed, let $\mathcal{P}'_{\mathcal{T}}(G)$ be a minimum kPC containing an available endpoint in $N[v_n]$, say v_e. Clearly, v_n belongs to path P containing more than one vertex; let $P = [v_n, v_k, \ldots]$. Removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$, and since $v_kv_e \in E$, results to a kPC of size $\lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction.

Suppose now that, when the algorithm processes vertex v_n, it inserts it into an existing path, and, thus, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$; consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum kPC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$. Furthermore, there exists no available endpoint in $N[v_n]$ in $\mathcal{P}_{\mathcal{T}}(G)$. Let $\mathcal{P}'_{\mathcal{T}}(G)$ be a minimum kPC of G having an available endpoint $v_\ell \in N[v_n]$, $i \leq \ell \leq n$. Clearly, v_n belongs to path P containing more than one vertex. If $v_\ell = v_n$, then removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum kPC of $G[S]$ having a neighbor of v_n, say v_t, as an available endpoint, a contradiction. If v_n is an internal node of a path, let

v_a and v_b be the vertices it is connected to. Clearly, $v_a v_b \in E(G)$ and, thus, removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum kPC of $G[S]$ having v_ℓ as an available endpoint, a contradiction.

Finally, suppose that, when the algorithm processes vertex v_n, it creates a new path, that is, $P_{\lambda_{\mathcal{T}}(G)} = [v_n]$, and, thus, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$. Since the algorithm creates a new path, there is no path in $\mathcal{P}_{\mathcal{T}}(G[S])$ having an available endpoint $v_a \in N(v_n)$ or containing an edge $v_a v_b$ such that $v_a, v_b \in N(v_n)$. Let $\mathcal{P}'_{\mathcal{T}}(G)$ be a minimum kPC of size $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$. If v_n is the only vertex of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a kPC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])-1$, a contradiction. If v_n is an endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum kPC of $G[S]$ having a neighbor of v_n, say v_t, as an available endpoint; a contradiction. On the other hand, if v_n is an internal node of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then it has at least two neighbors, that is $v_{n-1}, v_{n-2} \in N(v_n)$ and $v_{n-1}v_{n-2} \in E(G)$. According to Lemma 9.4, unless vertices v_{n-1} and v_{n-2} are terminal vertices, they are successive in a path in $\mathcal{P}_{\mathcal{T}}(G[S])$ and/or at least one of them is an available endpoint. Since the algorithm constructs a new path, $v_{n-1}, v_{n-2} \in \mathcal{T}$ and they are endpoints at the same path (see Lemma 9.3) which contains at least one more vertex, that is, v_{n-2} is the left endpoint of $P_{\lambda_{\mathcal{T}}(G[S])}$ and v_{n-1} is its right endpoint. Consequently, if we apply the algorithm to $G[S] - \{v_{n-1}, v_{n-2}\} = G - \{v_{n-1}, v_n, v_{n-2}\}$, we obtain a kPC of size $\lambda_{\mathcal{T}}(G[S])$, which, by the induction hypothesis, is minimum. Suppose that $N(v_n) = \{v_{n-1}, v_{n-2}\}$. Then, the minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ contains the path $P' = [v_{n-1}, v_n, v_{n-2}]$. Consequently, removing the vertices v_{n-1}, v_n, v_{n-2} from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a kPC of $G - \{v_{n-1}, v_n, v_{n-2}\}$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction. Now let $|N(v_n)| > 2$. In this case, according to Lemma 9.5, the algorithm either connects v_n to a path or inserts it into a path and thus, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$, a contradiction.

Consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum kPC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$. Finally, the rightmost available endpoint in $N[v_n]$ of the kPC constructed by the algorithm is vertex v_n and, clearly, it is the rightmost available endpoint in $N[v_n]$ that any other minimum kPC $\mathcal{P}'_{\mathcal{T}}(G)$ can contain. ∎

Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a terminal set. Then, Algorithm Minimum_kPC computes a minimum kPC $\mathcal{P}_{\mathcal{T}}(G)$ of G. Both time and space complexities are $O(n+m)$ since vertices p_ℓ and p_r of the algorithm can be maintained and obtained in constant time and the paths are implemented as linked lists. Furthermore, the canonical ordering is constructed in $O(n+m)$ time [64]. Hence, we can state the following result.

Theorem 9.3. *Let G be a proper interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$. A minimum k-fixed-endpoint path cover $\mathcal{P}_{\mathcal{T}}(G)$ of G can be computed in $O(n+m)$ time and space.*

9.5 Concluding Remarks

An interesting open question would be to see if the k-fixed-endpoint path cover problem can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. Recall that the complexity status of simpler problems, such as 1HP and 2HP, is unknown for interval graphs.

Chapter 10

The 1-fixed-endpoint Path Cover Problem is Polynomial on Interval Graphs

10.1 Introduction

We consider a variant of the path cover problem, namely, the k-fixed-endpoint path cover problem, or kPC for short, on interval graphs. Recall that, given a graph G and a subset $\mathcal{T}$ of k vertices of $V(G)$, a k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the k vertices of $\mathcal{T}$ are all endpoints of the paths in $\mathcal{P}$. The kPC problem is to find a k-fixed-endpoint path cover of G of minimum cardinality; note that, if $\mathcal{T}$ is empty the stated problem coincides with the classical path cover problem. In this chapter, we study the 1-fixed-endpoint path cover problem on interval graphs, or 1PC for short, generalizing the 1HP problem which has been proved to be NP-complete even for small classes of graphs. Motivated by a work of Damaschke [27], where he left both 1HP and 2HP problems open for the class of interval graphs, we show that the 1PC problem can be solved in polynomial time on the class of interval graphs. The proposed algorithm runs in $O(n^2)$ time, requires linear space, and also enables us to solve the 1HP problem on interval graphs within the same time and space complexity. We next define the 1PC problem.

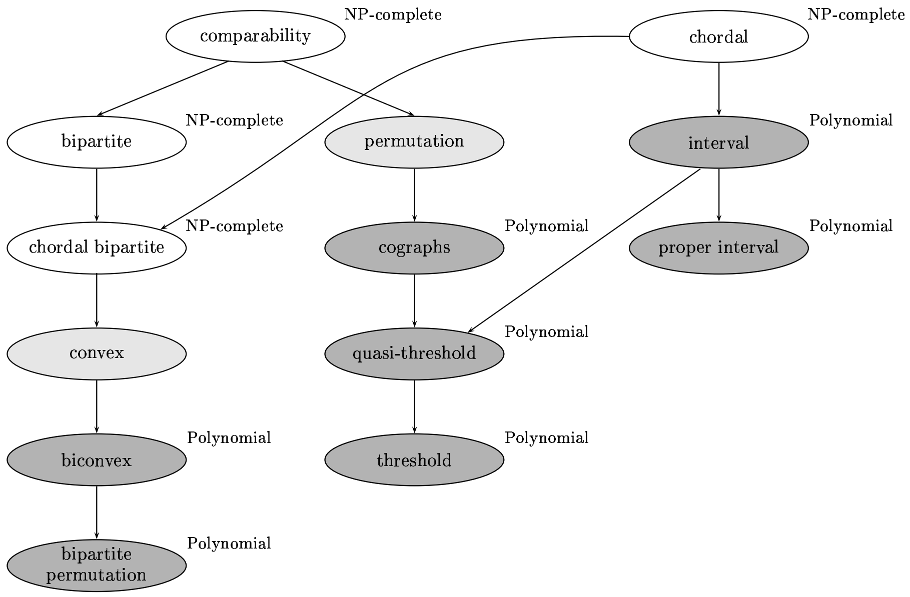


Figure 10.1: The complexity status (NP-complete, unknown, polynomial) of the 1HP problem for some graph subclasses of comparability and chordal graphs. $A \rightarrow B$ indicates that class A contains class B.

Problem 1PC. Given a graph G and a vertex $u \in V(G)$, a *1-fixed-endpoint path cover* of the graph G with respect to u is a path cover of G such that the vertex u is an endpoint of a path in the path cover; a *minimum 1-fixed-endpoint path cover* of G with respect to u is a 1-fixed-endpoint path cover of G with minimum cardinality; the *1-fixed-endpoint path cover problem* (1PC) is to find a minimum 1-fixed-endpoint path cover of the graph G.

Contribution. In this chapter, we study the complexity status of the 1-fixed-endpoint path cover problem (1PC) on the class of interval graphs [13, 40], and show that this problem can be solved in polynomial time. The proposed algorithm runs in $O(n^2)$ time on an interval graph G on n vertices and m edges and requires linear space. The proposed algorithm for the 1PC problem can also be used to solve the 1HP problem on interval graphs within the same time and space complexity. Using our algorithm for the 1PC problem and a simple reduction described by Müller in [69], we solve the HP problem on a X-convex graph $G = (X, Y, E)$ with $|Y| - |X| = 1$, which was left open in [91]. We also show that the 1HP problem on a convex graph G is solvable in time quadratic in the number of its vertices. Figure 10.1 shows a diagram of class inclusions for a number of graph classes, subclasses of comparability and chordal graphs, and the current complexity status of the 1HP problem on these classes; for definitions of the classes shown, see [13, 40].

Related Work. Interval graphs form an important class of perfect graphs [40] and many problems that are NP-complete on arbitrary graphs are shown to admit polynomial time algorithms on this class [4, 40, 58]. Both Hamiltonian Circuit (HC) and Hamiltonian Path (HP) problems are polynomially solvable for the class of interval and proper interval graphs. Keil introduced a linear-time algorithm for the HC problem on interval graphs [58] and Arikati and Rangan [4] presented a linear-time algorithm for the minimum path cover problem on interval graphs. Bertossi [8] proved that a proper interval graph has a Hamiltonian path if and only if it is connected. He also gave an $O(n \log n)$ algorithm for finding a Hamiltonian circuit in a proper interval graph. Furthermore, Lin et al. [61] proposed an optimal algorithm for the path cover problem on cographs while Nakano et al. [70] proposed an optimal parallel algorithm which finds and reports all the paths in a minimum path cover of a cograph in $O(\log n)$ time using $O(n/\log n)$ processors on a PRAM model. Hsieh et al. [49] presented an $O(n+m)$-time sequential algorithm for the Hamiltonian problem on a distance-hereditary graph and also proposed a parallel implementation of their algorithm which solves the problem in $O(\log n)$ time using $O((n+m)/\log n)$ processors on a PRAM model. A unified approach to solving the Hamiltonian problems on distance-hereditary graphs was presented in [50], while Hsieh [48] presented an efficient parallel strategy for the 2HP problem on the same class of graphs. Algorithms for the path cover problem on other classes of graphs were proposed in [51, 72, 86].

Road Map. The chapter is organized as follows. In Section 10.2 we establish the notation and related terminology, and we present background results. In Section 10.3 we describe our algorithm for the 1PC problem, while in Section 10.4 we prove its correctness and compute its time and space complexity. Section 10.5 presents some related results and in Section 10.6 we conclude the chapter and discuss possible future extensions.

10.2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges. For a graph G, we denote its vertex and edge set by $V(G)$ and $E(G)$, respectively. Let S be a subset of the vertex set of a graph G. Then, the subgraph of G induced by S is denoted by $G[S]$.

10.2.1 Structural Properties of Interval Graphs

A graph G is an *interval graph* if its vertices can be put in a one-to-one correspondence with a family F of intervals on the real line such that two vertices are adjacent in G if and only if their corresponding intervals intersect. F is called an *intersection model* for G [4]. Interval graphs find applications in genetics, molecular biology, archaeology, and storage information retrieval [40]. Interval graphs form an important class of perfect graphs [40] and many problems that are NP-complete on arbitrary graphs are shown to admit polynomial time algorithms on this class [4, 40, 58]. The class of interval graphs is *hereditary*, that is, every induced subgraph of an interval graph G is also an interval graph. We state the following numbering for the vertices of an interval graph proposed in [78].

Lemma 10.1. *(Ramalingam and Rangan [78]): The vertices of any interval graph G can be numbered with integers $1, \ldots, |V(G)|$ such that if $i < j < k$ and $ik \in E(G)$ then $jk \in E(G)$.*

As shown in [78], the numbering of Lemma 10.1, which results from numbering the intervals after sorting them on their right ends [4], can be obtained in linear time, that is, $O(m+n)$ time. An ordering of the vertices according to this numbering is found to be quite useful in solving many problems on interval graphs [4, 78]. Throughout the chapter, the vertex numbered with i will be denoted by v_i, $1 \leq i \leq n$, and such an ordering will be denoted by π. We say that $v_i < v_j$ if $i < j$, $1 \leq i, j \leq n$.

10.2.2 Interval Graphs and the 1PC Problem

Let G be an interval graph with vertex set $V(G)$ and edge set $E(G)$, $\mathcal{T}$ be a set containing a single vertex of $V(G)$, and let $\mathcal{P}_{\mathcal{T}}(G)$ be a minimum 1-fixed-endpoint path cover of G with respect to $\mathcal{T}$ of size $\lambda_{\mathcal{T}}(G)$ (or $\lambda_{\mathcal{T}}$ for short); recall that the size of $\mathcal{P}_{\mathcal{T}}(G)$ is the number of paths it contains. The vertex belonging to the set $\mathcal{T}$ is called *terminal* vertex, and the set $\mathcal{T}$ is called the *terminal set* of G, while those of $V(G) - \mathcal{T}$

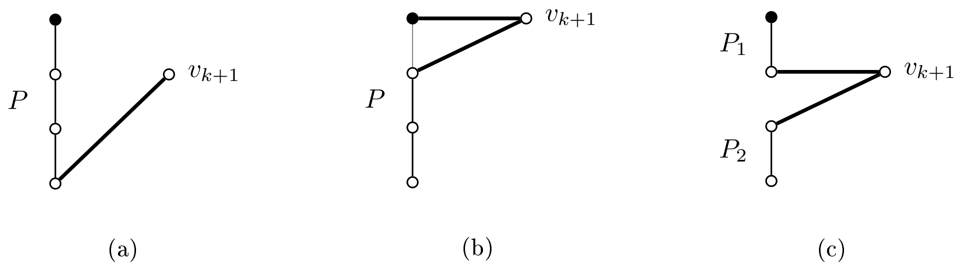


Figure 10.2: Illustrating (a) connect, (b) insert, and (c) bridge operations; $P, P_1, P_2 \in \mathcal{P}_{\mathcal{T}}(G[S])$.

are called *non-terminal or free* vertices. Thus, the set $\mathcal{P}_{\mathcal{T}}(G)$ contains two types of paths, which we call *terminal* and *non-terminal or free* paths: a *terminal path* P_t is a path having the terminal vertex as an endpoint and a *non-terminal or free path* P_f is a path having both its endpoints in $V(G) - \mathcal{T}$. The set of the non-terminal paths in a minimum 1PC of the graph G is denoted by N, while T denotes the set containing the terminal path. Clearly, $|T| = 1$ and $\lambda_{\mathcal{T}} = |N| + 1$.

Our algorithm for computing a 1PC of an interval graph is based on a greedy principle, visiting the vertices according to the ordering $\pi = (v_1, v_2, \ldots, v_k, \ldots, v_n)$, and uses three operations on the paths of a 1PC of $G[S]$, where $S = \{v_1, v_2, \ldots, v_k\}$, $1 \leq k < n$. These three operations, namely `connect`, `insert` and `bridge` operations, are described below and are illustrated in Fig. 10.2.

- `Connect` operation: Let v_i be a free endpoint of a path P of $\mathcal{P}_{\mathcal{T}}(G[S])$ and let v_{k+1} be a free or a terminal vertex such that v_{k+1} sees v_i. We say that we *connect* vertex v_{k+1} to the path P, or, equivalently, to the vertex v_i, if we extend the path P by adding an edge which joins vertex v_{k+1} with vertex v_i.
- `Insert` operation: Let $P = (\ldots, v_i, v_j, \ldots)$, $i \neq j$, $i, j \in [1, k]$, be a path of $\mathcal{P}_{\mathcal{T}}(G[S])$ and let v_{k+1} be a free vertex such that v_{k+1} sees v_i and v_j. We say that we *insert* vertex v_{k+1} into P, if we replace the path P with the path $P' = (\ldots, v_i, v_{k+1}, v_j, \ldots)$.
- `Bridge` operation: Let P_1 and P_2 be two paths of $\mathcal{P}_{\mathcal{T}}(G[S])$ and let v_{k+1} be a free vertex that sees at least one free endpoint of P_1 and at least one of P_2. We say that we *bridge* the two paths P_1 and P_2 using vertex v_{k+1} if we connect v_{k+1} with a free endpoint of P_1 and a free endpoint of P_2.

Let P be a path of $\mathcal{P}_{\mathcal{T}}(G)$ and let v_i and v_j be its endpoints. We say that v_i is the left (resp. right) endpoint of the path and v_j is the right (resp. left) endpoint of the path if $v_i < v_j$ (resp. $v_j < v_i$). Throughout the chapter, a trivial path (i.e. a path consisting of one vertex) is considered to have two endpoints, while a trivial path consisting of the terminal vertex $u \in \mathcal{T}$ is considered to have one terminal endpoint and one free endpoint.

Let G be an interval graph on n vertices and let $\mathcal{P}_{\mathcal{T}}(G)$ be a minimum 1PC of size $\lambda_{\mathcal{T}}$. Since a trivial path is considered to have two endpoints, the number of endpoints in $\mathcal{P}_{\mathcal{T}}(G)$ is $2\lambda_{\mathcal{T}}$. For each vertex v_i we denote by $d(v_i)$ the number of neighbors of v_i in $\mathcal{P}_{\mathcal{T}}(G)$; that is, $d(v_i) \in \{0, 1, 2\}$. We call *d-connectivity* of $\mathcal{P}_{\mathcal{T}}(G)$ the sum of $d(v_1), d(v_2), \ldots, d(v_n)$. It is easy to see that $\sum_{i=1}^{n} d(v_i) = 2(n - \lambda_{\mathcal{T}})$. Clearly, any minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ has d-connectivity equal to $2(n - \lambda_{\mathcal{T}})$.

10.3 The Algorithm

We next present an algorithm for the 1PC problem on interval graphs. Our algorithm takes as input an interval graph G on n vertices and m edges and a set $\mathcal{T} = \{u\}$ containing the terminal vertex $u \in V(G)$, and computes a minimum 1PC $\mathcal{P}_\mathcal{T}(G)$ of G in $O(n^2)$ time; it is based on a greedy principle to extend a path of a minimum 1PC using operations on the left and right endpoints of its paths and properties of the graph $G[\{v_1, v_2, \ldots, v_i\} - \{u\}]$, $1 \leq i \leq n$. We point out that, if a vertex sees the two endpoints of only one non-terminal path P, it is connected to the left endpoint of the path P. Furthermore, for each vertex v_i, $1 \leq i < j$, we denote by $\varepsilon_i^{(j)}$ the number of endpoints v_κ belonging to different paths in $\mathcal{P}_\mathcal{T}(G[v_1, \ldots, v_j])$ with index $\kappa \in (i, j]$. We also define $\varepsilon_i^{(i)} = 0$ and $\varepsilon_0^{(i)} = \lambda_\mathcal{T}(G[v_1, \ldots, v_i])$, $1 \leq i \leq n$.

Before describing our algorithm, we show that, if $\lambda_\mathcal{T}(G)$ is the size of a minimum 1PC of G with respect to $\mathcal{T} = \{v_t\}$ then the size of a minimum 1PC of $G - \{v_t\}$ is either $\lambda_\mathcal{T}(G)$ or $\lambda_\mathcal{T}(G) - 1$.

Lemma 10.2. *Let G be an interval graph and $\lambda_\mathcal{T}(G)$ be the size of a minimum 1PC of G with respect to $\mathcal{T} = \{v_t\}$. The size of a minimum PC of $G - \{v_t\}$ is either $\lambda_\mathcal{T}(G)$ or $\lambda_\mathcal{T}(G) - 1$.*

Proof. Suppose that the size of a minimum PC of $G - \{v_t\}$ is at least $\lambda_\mathcal{T}(G) + 1$. Since a terminal vertex cannot decrease the size of a minimum 1PC, we have $\lambda_\mathcal{T}(G) \geq \lambda_\mathcal{T}(G - \{v_t\})$. Thus, $\lambda_\mathcal{T}(G) \geq \lambda_\mathcal{T}(G) + 1$, a contradiction. Suppose now that the size of a minimum PC $\mathcal{P}_\mathcal{T}(G - \{v_t\})$ of $G - \{v_t\}$ is at most $\lambda_\mathcal{T}(G) - 2$. Then, adding a trivial path containing vertex v_t to $\mathcal{P}_\mathcal{T}(G - \{v_t\})$ results to a 1PC of G of size $\lambda_\mathcal{T}(G) - 1$, a contradiction. ∎

The algorithm works as follows:

Algorithm Minimum_1PC

Input: an interval graph G on n vertices and m edges and a vertex $u \in V(G)$;

Output: a minimum 1PC $\mathcal{P}_\mathcal{T}(G)$ of the interval graph G;

```
1. Construct the ordering $\pi$ of the vertices of $G$;

2. Execute the subroutine $process(\pi)$; the minimum 1PC $\mathcal{P}_\mathcal{T}(G)$ is the set of paths returned by the
   subroutine;
```

Algorithm 8: Algorithm Minimum_1PC

where the description of the subroutine *process*(π) is presented below.

```
*process* ($\pi$)
*Input:* the ordering $\pi$ of the vertices of $G$ and the index $t$ of the terminal vertex $u$;
*Output:* a minimum 1PC $\mathcal{P}_T$ of $G$;
$\lambda_T = 1$; $P_{\lambda_T} = (v_1)$; $\varepsilon_0^{(1)} = 1$;
$p_1^\ell = v_1$; $p_1^r = v_1$;        {*the left and right endpoints of the path* $P_{\lambda_T}$}
for $i = 2$ to $n$ do
▷ if $i \neq t$ then
  ○ if $N'(v_i) \neq \emptyset$ and $\varepsilon_{j-1}^{(i-1)} \geq 2$ then {$N'(v_i)$={$v_j \in N(v_i) : j < i$}, $v_j$ *is the leftmost neighbor of* $v_i \in N'(v_i)$}
      if at least two endpoints are free vertices then bridge; $\lambda_T = \lambda_T - 1$;
      else   {$\varepsilon_{j-1}^{(i-1)} = 2$ *and one endpoint is the terminal vertex* $v_t$*, and the other, say,* $v_f$*, is a free vertex.*}
        if process({$v_1, \ldots, v_{i-1}$} − {$v_t$}) returns a 1PC $\mathcal{P}_T(G[\{v_1, \ldots, v_{i-1}\} - \{v_t\}])$ of $\lambda_T - 1$ paths
        then
           connect $v_i$ to the leftmost endpoint of $\mathcal{P}_T(G[\{v_1, \ldots, v_{i-1}\} - \{v_t\}])$; connect $v_t$ to $v_i$;
           $\lambda_T = \lambda_T - 1$;
        else-if $flag = 0$ and process({$v_1, \ldots, v_{i-1}$} − {$v_t$}) returns a 1PC of $\lambda_T$ paths
        then
          $flag = 1$;
          if process({$v_1, \ldots, v_i$}) returns a 1PC $\mathcal{P}_T(G[\{v_1, \ldots, v_i\}])$ of $\lambda_T - 1$ paths then
             $flag = 0$; $\lambda_T = \lambda_T - 1$;
        else connect_break;
  ○ if $N'(v_i) \neq \emptyset$ and $\varepsilon_{j-1}^{(i-1)} = 1$ and the endpoint $v_f, j \leq f \leq i-1$, is a free vertex then
      if $v_i$ sees an internal vertex $v_s$ then connect_break;
      else connect $v_i$ to the leftmost free endpoint;
  ○ if $N'(v_i) = \emptyset$ or $\varepsilon_{j-1}^{(i-1)} = 0$ or ($\varepsilon_{j-1}^{(i-1)} = 1$ and the endpoint $v_t, j \leq t \leq i-1$, is the terminal vertex)
    then
      if $v_i$ has two consecutive neighbors into a path then insert $v_i$ into the path;
      else-if $v_i$ sees an internal vertex $v_s$ then
        if $v_s v_a$ is an edge of a path $P_k$ and $v_a$ sees an endpoint $v_b$ of a path $P_{k'}, k \neq k'$ then
           remove the edge $v_s v_a$ of $P$; connect $v_a$ to $v_b$; connect $v_i$ to $v_j$;
        else new_path; $\lambda_T = \lambda_T + 1$;
      else-if $v_t \in N(v_i)$ and $flag = 1$ then
        break $v_t$ from its path; connect $v_i$ to $v_t$; $\lambda_T = \lambda_T + 1$;
      else $\lambda_T = \lambda_T + 1$; $P_{\lambda_T} = (v_i)$;
  endif;
▷ if $i = t$ then
    if $\varepsilon_{j-1}^{(i-1)} \geq 1$ then connect $v_i$ to the leftmost endpoint of $\mathcal{P}_T(G[\{v_1, \ldots, v_{i-1}\}])$;
    else $\lambda_T = \lambda_T + 1$; $P_{\lambda_T} = (v_i)$;
  endif;
  for $k = 0$ to $i$ do update $\varepsilon_k^{(i)}$; update_endpoints;
endfor;
$\mathcal{P}_T(G) = \{P_1, \ldots, P_{\lambda_T}\}$.
```

We next describe the operation `bridge` in detail. Note that in most cases we bridge two paths that have the leftmost free endpoints. Suppose that when vertex v_i is processed it sees at least one free endpoint of a non-terminal path P_1, say, v_j, and at least the free endpoint of the terminal path P_2, say, v_ℓ, and both endpoints of a non-terminal path P_3, say, v_r and v_s. Let v_z, v_t and v_r be the left endpoints and v_j, v_ℓ and v_s be the right endpoints of the paths P_1, P_2 and P_3, respectively. There exist three

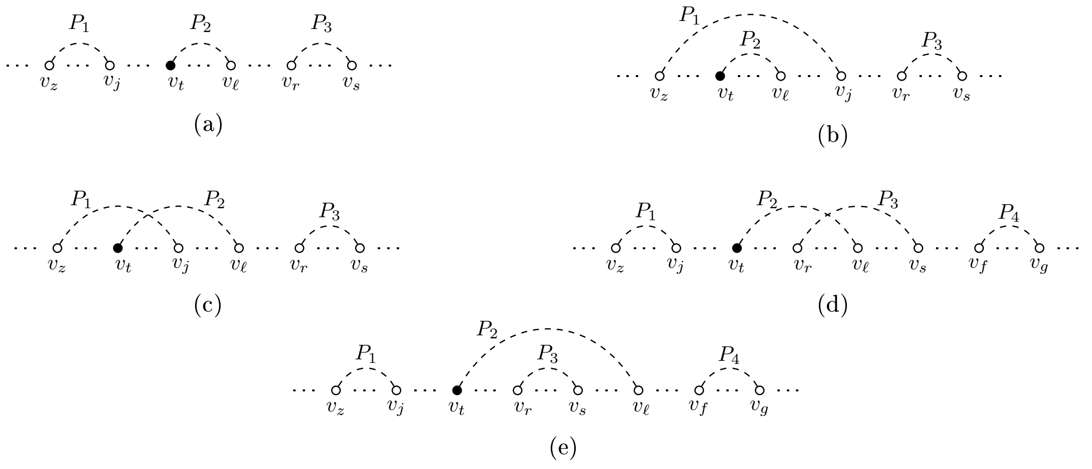


Figure 10.3: Illustrating some cases of the bridge operation.

cases where, in order to obtain the maximum possible value for every $\varepsilon_i^{(j)}$, we do not bridge two paths through the leftmost free endpoints (see Fig. 10.3(a)-(c)). In these three cases the bridge operation works as follows:

(a) $v_z < v_j < v_t < v_\ell < v_r < v_s$: we bridge P_1 and P_3 through v_z (or, v_j if $v_z \notin N(v_i)$) and v_r.

(b) $v_z < v_t < v_\ell < v_j < v_r < v_s$: if $v_z \notin N(v_i)$ we bridge P_2 and P_3 through v_ℓ and v_r; otherwise, we bridge P_1 and P_2 through v_z and v_ℓ.

(c) $v_z < v_t < v_j < v_\ell < v_r < v_s$: we bridge P_1 and P_3 through v_z (or, v_j if $v_z \notin N(v_i)$) and v_r.

Suppose now that P_1 is a non-terminal path having v_z and v_j as its left and right endpoints, respectively, P_2 is the terminal path with left endpoint v_t and right endpoint v_ℓ. Also, let P_3 be a non-terminal path with left and right endpoints v_r and v_s, respectively, and P_4 a non-terminal path with left and right endpoints v_f and v_g, respectively (see Fig. 10.3(d)-(e)). We distinguish the following two cases:

(d) $v_z < v_j < v_t < v_r < v_\ell < v_s < v_f < v_g$: if $v_z \in N(v_i)$ or $v_j \in N(v_i)$ we bridge paths P_1 and P_3 through v_z (v_j if $v_z \notin N(v_i)$) and v_r. If $v_\ell \in N(v_i)$ and $v_r \notin N(v_i)$ we bridge P_2 and P_4 through v_ℓ and v_f.

(e) $v_z < v_j < v_t < v_r < v_s < v_\ell < v_f < v_g$: if $v_z \in N(v_i)$ or $v_j \in N(v_i)$ we bridge paths P_1 and P_3 through v_i (v_j if $v_z \notin N(v_i)$) and v_r; otherwise, we bridge P_3 and P_4 through v_r (v_s if $v_r \notin N(v_i)$) and v_f.

Figure 10.3 presents cases (a)-(e). Suppose that we have the two paths P_2 and P_3 of case (e) and vertex v_i sees both v_r and v_s, that is, $P_2 = (v_t, \ldots, v_a, v_b, v_c, \ldots, v_\ell)$ and $P_3 = (v_r, \ldots, v_s)$, where $v_a < v_s < v_b$ and $v_s < v_c$. Then, the bridge operation constructs the path $P = (v_t, \ldots, v_a, v_b, v_s, \ldots, v_r, v_i, v_c, \ldots, v_\ell)$. Suppose now that we have the two paths P_1 and P_2 of case (c) and vertex v_i sees all vertices with index greater or equal to z, that is, $P_1 = (v_z, \ldots, v_j)$ and $P_2 = (v_t, \ldots, v_a, v_b, v_c, \ldots, v_\ell)$, where $v_a < v_j < v_b$ and $v_j < v_c$. Then, the bridge operation constructs the path $P = (v_t, \ldots, v_a, v_b, v_j, \ldots, v_z, v_i, v_c, \ldots, v_\ell)$. Suppose that there exist two paths P_2 and P_3 as in case (d) and vertex v_i sees all vertices with index k, $d \leq k$, where $r < d \leq \ell$, that is, $P_2 = (v_t, \ldots, v_a, v_b, v_c, \ldots, v_\ell)$ and $P_3 = (v_r, \ldots, v_s)$, where $v_a < v_r < v_b$ and $v_r < v_c$.

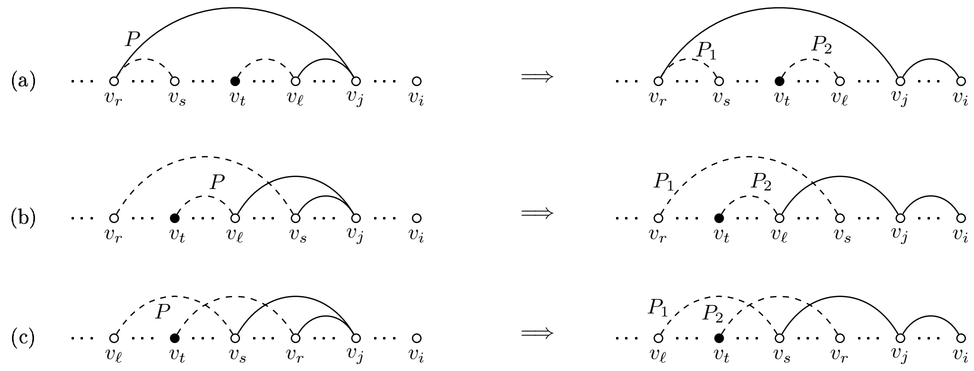


Figure 10.4: Illustrating some cases of the new_path operation.

If $d < c$ then the bridge operation constructs the path $P = (v_t, \ldots, v_a, v_b, v_r, \ldots, v_s, v_i, v_c, \ldots, v_\ell)$; otherwise, it constructs the path $P = (v_t, \ldots, v_a, v_b, v_r, \ldots, v_s, v_i, v_\ell, \ldots, v_c)$. If there exist two paths P_1 and P_2 as in case (b) and vertex v_i sees all vertices with index greater or equal to z, that is, $P_1 = (v_z, \ldots, v_a, v_b, v_c, \ldots, v_j)$ and $P_2 = (v_t, \ldots, v_\ell)$, where $v_a < v_\ell < v_b$ and $v_\ell < v_c$, then the bridge operation constructs the path $P = (v_t, \ldots, v_\ell, v_b, v_a, \ldots, v_z, v_i, v_c, \ldots, v_j)$, if $c < j$, or the path $P = (v_t, \ldots, v_\ell, v_b, v_a, \ldots, v_z, v_i, v_j, \ldots, v_c)$, if $j < c$.

We next describe the operation `new_path` which creates a new path when the vertex v_i is processed. There exist three cases where operation new_path creates a new non-trivial path while in all other cases it creates a new trivial path. Suppose that v_i sees an internal vertex v_j belonging to a path $P = (v_s, \ldots, v_r, v_j, v_\ell, \ldots, v_t)$ such that $v_r < v_s < v_t < v_\ell < v_j$. We remove the edge $v_j v_\ell$ from P and we obtain $P_1 = (v_s, \ldots, v_r, v_j)$ and $P_2 = (v_t, \ldots, v_\ell)$. Then, we connect v_i to v_j. The case where $v_j v_s \in E(G)$ and $v_j v_r \notin E(G)$ is similar. If v_i sees an internal vertex v_j belonging to a path $P = (v_r, \ldots, v_s, v_j, v_\ell, \ldots, v_t)$ such that $v_r < v_t < v_\ell < v_s < v_j$, we remove the edge $v_s v_j$ from P and we obtain $P_1 = (v_r, \ldots, v_s)$ and $P_2 = (v_t, \ldots, v_\ell, v_j)$. Then, we connect v_i to v_j. Suppose now that v_i sees an internal vertex v_j belonging to a path $P = (v_\ell, \ldots, v_s, v_j, v_r, \ldots, v_t)$ such that $v_\ell < v_t < v_s < v_r < v_j$. We remove the edge $v_j v_r$ from P and we obtain $P_1 = (v_\ell, \ldots, v_s, v_j)$ and $P_2 = (v_t, \ldots, v_r)$. Then, we connect v_i to v_j. The above cases, where the operation new_path creates a new non-trivial path, are described below:

(a) $v_r < v_s < v_t < v_\ell < v_j$: We create paths $P_1 = (v_s, \ldots, v_r, v_j, v_i)$ and $P_2 = (v_t, \ldots, v_\ell)$. The case where $v_j v_s \in E(G)$ and $v_j v_r \notin E(G)$ is similar.

(b) $v_r < v_t < v_\ell < v_s < v_j$: We create paths $P_1 = (v_r, \ldots, v_s)$ and $P_2 = (v_t, \ldots, v_\ell, v_j, v_i)$.

(c) $v_\ell < v_t < v_s < v_r < v_j$: We create paths $P_1 = (v_\ell, \ldots, v_s, v_j, v_i)$ and $P_2 = (v_t, \ldots, v_r)$.

Note that, the rightmost endpoint of a path in $\mathcal{P}_\mathcal{T}(G[\{v_1, \ldots, v_{i-1}\}])$ is v_t and, thus, $\varepsilon_t^{(i-1)} = 0$. The 1PC $\mathcal{P}_\mathcal{T}(G)$ of the graph G in each of the above cases contains two new endpoints, vertices v_i and $v_{k'}$ such that $t < k'$; thus, $\varepsilon_t^{(i)} = 2$. Figure 10.4 presents the above cases.

The operation `connect_break` is similar to the operation new_path. Specifically, suppose that in the above cases (a)-(c) there exists a path $P = (v_a, \ldots, v_b)$ such that $v_j < v_a < v_b < v_i$. Then, the operation connect_break works similarly to the operation new_path; the only difference is that v_i is also connected to v_a.

Concerning the ordering of the endpoints of the paths of the 1PC constructed by Algorithm Minimum_1PC, we prove the following lemma.

Lemma 10.3. *Let G be an interval graph with no terminal vertex. Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$, be a path in the 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G constructed by Algorithm Minimum_1PC and let v_i and v_j be the left and right endpoints of P_s, respectively. Then, there is no path $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, such that $v_i < v_t < v_j$ or $v_i < v_\ell < v_j$, where v_k and v_ℓ are the left and right endpoints of P_t, respectively.*

Proof. Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$ be a path in the 1PC $\mathcal{P}_{\mathcal{T}}(G)$ constructed by Algorithm Minimum_1PC and let v_i and v_j be its left and right endpoints, respectively. Let $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, and let v_k and v_ℓ be its left and right endpoints, respectively. Suppose that $v_i < v_k < v_j$. Since v_i and v_j are the endpoints of P_s and $v_i < v_k < v_j$, the path P_s contains at least one edge, say, $v_a v_b$, such that $v_a < v_k < v_b$. Clearly, vertices v_a and v_b are free vertices. Since $v_a v_b \in E(G)$, we also have $v_k v_b \in E(G)$. Then, according to Algorithm Minimum_1PC, when vertex v_b is processed, vertex v_k is an endpoint of the path P_t, and, thus, v_b bridges the paths P_s and P_t through vertices v_a and v_k, a contradiction. Similarly, we can prove that $v_\ell < v_i$ or $v_\ell > v_j$. ∎

Using similar arguments we can prove the following lemma:

Lemma 10.4. *Let G be an interval graph containing a terminal vertex. Let P_s, $1 \leq s \leq \lambda_{\mathcal{T}}$, be a non-terminal path in the 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G constructed by Algorithm Minimum_1PC and let v_i and v_j be the left and right endpoints of P_s, respectively. Then, there is no non-terminal path $P_t \in \mathcal{P}_{\mathcal{T}}(G)$, $1 \leq t \leq \lambda_{\mathcal{T}}$, $t \neq s$, such that $v_i < v_k < v_j$ or $v_i < v_\ell < v_j$, where v_k and v_ℓ are the left and right endpoints of P_t, respectively.*

10.4 Correctness and Time Complexity

Let G be an interval graph on n vertices and m edges and let $\mathcal{T}$ be a subset of $V(G)$ containing a single vertex. In order to prove the correctness of Algorithm Minimum_1PC, we use induction on n. We also prove a property of the minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of G constructed by our algorithm: Algorithm Minimum_1PC computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i < \varrho$, where ϱ is the index of the rightmost endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$. Furthermore, one of the following holds:
(i) $\varepsilon_i'^{(n)} \leq \varepsilon_i^{(n)}$, $\varrho \leq i \leq n$, or
(ii) if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$, where ϱ' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.
Recall that a trivial path has two endpoints that coincide. Hence, we prove the following theorem.

Theorem 10.1. *Let G be an interval graph on n vertices and m edges and let $v \in V(G)$. Algorithm Minimum_1PC computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i < \varrho$, where ϱ is the index of the rightmost endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$. Furthermore, one of the following holds:*

(i) $\varepsilon_i'^{(n)} \leq \varepsilon_i^{(n)}$, $\varrho \leq i \leq n$, or

(ii) if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$, where ϱ' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.

Proof. We use induction on n. The basis $n = 1$ is trivial. Assume that Algorithm Minimum_1PC computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G[S])$ of every interval graph $G[S]$, $S = \{v_1, v_2, \ldots, v_{n-1}\}$, with at most $n-1$ vertices having $\varepsilon_i^{(n-1)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n-1]$, $1 \leq i \leq n-1$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G[S])$ having $\varepsilon_i'^{(n-1)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n-1]$ such that $\varepsilon_i'^{(n-1)} > \varepsilon_i^{(n-1)}$, $1 \leq i < d$, where d is the index of the rightmost endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$. Furthermore, one of the following holds:

(i) $\varepsilon_i'^{(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$, or

(ii) if $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)} + 1$, $d \leq i < d'$ and $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)}$, $d' \leq i \leq n-1$, where d' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G[S])$, then there exists a vertex v_q such that $\varepsilon_q^{(n-1)} > \varepsilon_q'^{(n-1)}$, $1 \leq q < d$ and there exists no vertex $v_{q'}$, $1 \leq q' < d$, such that $\varepsilon_{q'}'^{(n-1)} > \varepsilon_{q'}^{(n-1)}$.

Let $\lambda_{\mathcal{T}}(G[S])$ be the size of $\mathcal{P}_{\mathcal{T}}(G[S])$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i < \varrho$, where ϱ is the index of the rightmost endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$. Furthermore, one of the following holds:

(i) $\varepsilon_i'^{(n)} \leq \varepsilon_i^{(n)}$, $\varrho \leq i \leq n$, or

(ii) if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$, where ϱ' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.

Case A: Vertex v_n is not the terminal vertex. Let $\lambda_{\mathcal{T}}(G)$ be the size of $\mathcal{P}_{\mathcal{T}}(G)$. Clearly, the size $\lambda'_{\mathcal{T}}(G)$ of a minimum 1PC of G is equal to $\lambda_{\mathcal{T}}(G[S]) - 1$ or $\lambda_{\mathcal{T}}(G[S])$ or $\lambda_{\mathcal{T}}(G[S]) + 1$. We distinguish the following cases:

***Case A.1*:** When the algorithm processes vertex v_n, it uses v_n to bridge two paths (operation bridge), that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) - 1$; consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$.

Case A.1.a: Suppose that $\varepsilon_i'^{(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i \leq n$. Clearly, vertex v_n is an internal vertex of a path in any other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$, otherwise removing it from $\mathcal{P}'_{\mathcal{T}}(G)$ would result to a 1PC of $G[S]$ of size $\leq \lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction. Assume that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$ such that $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$. Suppose that $\varepsilon_{k-1}'^{(n)} - \varepsilon_{k-1}^{(n)} = 1$. Note that $\varepsilon_1^{(n)} \geq \varepsilon_1'^{(n)}$. Indeed, let $\mathcal{P}'_{\mathcal{T}}(G)$ be a minimum 1PC of G having $\varepsilon_1'^{(n)} = \varepsilon_1^{(n)} + 1$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' > 1$. Suppose that vertex v_1 is an internal vertex in $\mathcal{P}_{\mathcal{T}}(G)$. Then, the algorithm constructs $\varepsilon_1^{(n)}$ paths while $\mathcal{P}'_{\mathcal{T}}(G)$ contains at least $\varepsilon_1^{(n)} + 1$ paths, a contradiction. Suppose that vertex v_1 is an endpoint in $\mathcal{P}_{\mathcal{T}}(G)$. Note that, according to the algorithm, if v_1 has degree greater or equal to one, then it belongs to a path containing more than one vertex. Consequently, the other endpoint of the path containing v_1 is one of the $\varepsilon_1^{(n)}$ endpoints, and, thus, the algorithm constructs $\varepsilon_1^{(n)}$ paths while $\mathcal{P}'_{\mathcal{T}}(G)$ contains at least $\varepsilon_1^{(n)} + 1$ paths, a contradiction. Since $\varepsilon_1^{(n)} \geq \varepsilon_1'^{(n)}$ there exists a vertex v_j,

$1 \le j < k-1$, such that $\varepsilon_j^{(n)} = \varepsilon_j'^{(n)}$. This implies that vertex v_{j+1} is the right endpoint of a path P in the minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ constructed by the algorithm. Without loss of generality, we assume that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $j+1 \le i < k-1$.

Let $P' = (\ldots, v_a, v_n, v_b, \ldots)$ be the path of $\mathcal{P}'_{\mathcal{T}}(G)$ containing vertex v_n. Then, $j+1 < a$ and $j+1 < b$. Indeed, suppose that at least one of v_a and v_b has index less or equal to $j+1$, say, $a < j+1$. Since $v_a v_n \in E(G)$ we have $v_{j+1}v_n \in E(G)$. Let $P = (\ldots, v_c, v_n, v_d, \ldots)$ be the path of $\mathcal{P}_{\mathcal{T}}(G)$ containing vertex v_n. Suppose that $v_c < v_{j+1}$ and $v_d < v_{j+1}$. Then, due to the induction hypothesis, both of the endpoints of P have index greater than $j+1$. However, according to the algorithm (operation bridge), such an ordering of the endpoints cannot exist, a contradiction. Suppose that $v_c > v_{j+1}$ and $v_d > v_{j+1}$. Then, due to the induction hypothesis, at least one of the endpoints of P should have index greater than $j+1$. Again, according to the algorithm (operation bridge), such an ordering of the endpoints cannot exist, a contradiction. Suppose now that $v_c < v_{j+1}$ and $v_d > v_{j+1}$. Then, due to the induction hypothesis, computing a 1PC of $G[S]$ we had case (e) which is described in Section 3 and $v_{j+1} = v_s$. However, in this case, vertex v_{j+1} would not be the right endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G[S])$, a contradiction.

Consequently, $j+1 < a$ and $j+1 < b$. Then, both endpoints of P have indexes less than $j+1$. This implies that vertex v_n has bridged two non-terminal paths for which we have an endpoint of one path between the endpoints of the other, which is a contradiction according to Lemma 10.4.

Consequently, we have shown that there does not exist a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$.

Case 1.b: Suppose that $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)} + 1$, $d \le i < d'$ and $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)}$, $d' \le i \le n-1$, where d' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G[S])$, and there exists a vertex v_q such that $\varepsilon_q^{(n-1)} > \varepsilon_q'^{(n-1)}$, $1 \le q < d$ and there exists no vertex v'_q, $1 \le q' < d$, such that $\varepsilon_q'^{(n-1)} > \varepsilon_q^{(n-1)}$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \le i \le n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$, $1 \le i \le \varrho$, such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \le i < \varrho$, where ϱ is the index of the rightmost endpoint of a path in $\mathcal{P}_{\mathcal{T}}(G)$. Furthermore, one of the following holds:

(i) $\varepsilon_i'^{(n)} \le \varepsilon_i^{(n)}$, $\varrho \le i \le n$, or

(ii) if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \le i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \le i \le n$, where ϱ' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$, then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \le z < \varrho$ and there exists no vertex v'_z, $1 \le z' < \varrho$, such that $\varepsilon_z'^{(n)} > \varepsilon_z^{(n)}$.

Suppose that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $\varepsilon_\varrho^{(n)} = \varepsilon_\varrho'^{(n)} = 0$. Then, similarly to Case A.1.a, we show that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \le i < n$.

Suppose now that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \le i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \le i \le n$, where ϱ' is the index of the rightmost endpoint of a path in $\mathcal{P}'_{\mathcal{T}}(G)$. We show that there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \le z < \varrho$ and there exists no vertex $v_{z'}$, $1 \le z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.

Note that, there cannot exist a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 2$, $\varrho \le i < \varrho'$. Indeed, suppose that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 2$, $\varrho \le i < \varrho'$. Consider the case where v_n belongs to a path in $\mathcal{P}'_{\mathcal{T}}(G)$ and it is connected to vertices $v_{a'}$ and $v_{b'}$ such that $\varrho < a'$ and $\varrho < b'$. Then, removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ we obtain a minimum 1PC of $G[S]$ such that $\varepsilon_\varrho'^{(n-1)} = \varepsilon_\varrho'^{(n)} + 1 = \varepsilon_\varrho^{(n)} + 3$ or $\varepsilon_\varrho'^{(n-1)} = \varepsilon_\varrho'^{(n)} + 2 = \varepsilon_\varrho^{(n)} + 4$. If v_n belongs to a path in $\mathcal{P}_{\mathcal{T}}(G)$ and it is connected to vertices v_a and v_b, then removing v_n from $\mathcal{P}_{\mathcal{T}}(G)$ we obtain a minimum 1PC of $G[S]$ such that $\varepsilon_\varrho^{(n-1)} \le \varepsilon_\varrho^{(n)} + 2$. Thus, there exist three paths in $\mathcal{P}_{\mathcal{T}}(G)$ such that there are no left endpoints between their right endpoints in π, a contradiction. Consider now the case where v_n belongs to a path in $\mathcal{P}'_{\mathcal{T}}(G)$ and it is connected to vertices $v_{a'}$ and $v_{b'}$ such that $\varrho < a'$ and $\varrho > b'$. Then, $v_n v_\varrho \in E(G)$. Note that v_n belongs to a path P in $\mathcal{P}_{\mathcal{T}}(G)$ and it is connected to vertices v_a and v_b such that $a < \varrho$ and $\varrho < b$. Removing v_n from $\mathcal{P}_{\mathcal{T}}(G)$ should result to a minimum 1PC of $G[S]$ such that $\varepsilon_\varrho^{(n-1)} = \varepsilon_\varrho^{(n)} + 1$. Consequently, both endpoints of P have index less than ϱ, which implies that there exists a specific ordering of the endpoints of the paths in $\mathcal{P}_{\mathcal{T}}(G[S])$, which, according to the algorithm, is not possible, a contradiction. The case where v_n belongs to a path in $\mathcal{P}'_{\mathcal{T}}(G)$ and it is connected to vertices $v_{a'}$ and $v_{b'}$ such that $\varrho > a'$ and $\varrho > b'$ is similar.

Using similar arguments with Case A.1.a, we show that there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}^{\prime(n)} > \varepsilon_{z'}^{(n)}$. Suppose that $\varepsilon_i^{(n)} = \varepsilon_i^{\prime(n)}$, $1 \leq i < \varrho$. Note that v_n belongs to a path $P = (v_\ell, \ldots, v_a, v_n, v_b, \ldots, v_r) \in \mathcal{P}_{\mathcal{T}}(G)$ such that $a > \varrho$ and $b > \varrho$. Thus, v_n belongs to a path $P' = (v_{\ell'}, \ldots, v_{a'}, v_n, v_{b'}, \ldots, v_{r'}) \in \mathcal{P}'_{\mathcal{T}}(G)$ such that $a' > \varrho$ and $b' > \varrho$. If we remove v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ then there exists a vertex $v_z = v_{b'-1}$ such that $\varepsilon_z^{\prime(n-1)} = \varepsilon_z^{\prime(n)} + 1$. This implies that $r' > b'$. Furthermore, if we remove v_n from $\mathcal{P}_{\mathcal{T}}(G)$ and we obtain $\varepsilon_z^{(n-1)} = \varepsilon_z^{(n)} + 1$, which implies that $b = b'$, then $\varepsilon_i^{(n-1)} = \varepsilon_i^{\prime(n-1)}$, $1 \leq i < \varrho$, a contradiction. If we remove v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ and we obtain $\varepsilon_z^{(n-1)} = \varepsilon_z^{(n)} + 2$, then there exists a specific ordering of the endpoints of the paths in $\mathcal{P}_{\mathcal{T}}(G[S])$ which, according to the algorithm, is not possible, a contradiction. Consequently, there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z^{\prime(n)}$, $1 \leq z < \varrho$.

***Case A.2*:** When the algorithm processes vertex v_n, it constructs a 1PC of size $\lambda_{\mathcal{T}}(G[S]) - 1$ of $\mathcal{P}_{\mathcal{T}}(G[S] - v_t)$ of $G[S] - v_t$, where v_t is the terminal vertex, and then connects the path (v_t, v_n) to an existing path. This operation is performed when vertex v_n sees the endpoints of at least one non-terminal path, say $P_1 = (v_r, \ldots, v_s)$, the terminal vertex v_t and no other endpoint of the terminal path, say, $P_2 = (v_t, \ldots, v_\ell)$. Then, the terminal path, P_2, has the same endpoints as in $\mathcal{P}_{\mathcal{T}}(G[S])$, the vertices of P_1 become internal vertices of P_2, while all the other paths in $\mathcal{P}_{\mathcal{T}}(G)$ remain the same as in $\mathcal{P}_{\mathcal{T}}(G[S])$. It is easy to see that there exists a path P in $\mathcal{P}_{\mathcal{T}}(G[S] - v_t)$ having an endpoint, say, $v_{s'}$, with index greater than t, and thus, $v_{s'} \in N(v_n)$. Note that when the connect operation is performed, it may use a vertex of the terminal path in order to increase the value of an $\varepsilon_i^{(n-1)}$, $1 \leq i < n-1$, and, in this case, $\varepsilon_d^{(n-1)} < \varepsilon_d^{\prime(n-1)}$. Then, for the endpoints of the terminal path, say, $v_t \in \mathcal{T}$ and v_ℓ, we have $v_t < v_\ell$. Consequently, since vertex v_n sees the endpoints of P_1, the terminal vertex v_t and no other endpoint of the terminal path P_2, if operation connect was called previously, it cannot have used a vertex of the terminal path and, thus, $\varepsilon_d^{(n)} < \varepsilon_d^{\prime(n)}$ cannot hold. Consequently, $\varepsilon_i^{\prime(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$.

The above procedure results to a 1PC of G of size $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) - 1$; consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$.

We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i^{\prime(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i^{\prime(n)} > \varepsilon_i^{(n)}$, $1 \leq i \leq n$. Clearly, vertex v_n is an internal vertex of a path in any other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$, otherwise removing it from $\mathcal{P}'_{\mathcal{T}}(G)$ would result to a 1PC of $G[S]$ of size less or equal to $\lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction. Suppose that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, such that $\varepsilon_{k-1}^{\prime(n)} > \varepsilon_{k-1}^{(n)}$ and $1 \leq k-1 < t-1$. This implies that $\varepsilon_k^{\prime(n)} = \varepsilon_k^{(n)}$ and there exists a vertex $v_{k'-1}$ such that $k'-1 < k-1$ and $\varepsilon_{k'-1}^{\prime(n)} = \varepsilon_{k'-1}^{(n)}$. Clearly, $v_{k'} v_n \notin E(G)$. Removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a minimum 1PC of $G[S]$ having at least two free neighbors of v_n as endpoints belonging to different paths, say, v_f and v_g; suppose that at least one of them has index greater than k. Then, $\varepsilon_{k-1}^{\prime(n-1)} > \varepsilon_{k-1}^{(n-1)}$, a contradiction. Thus, $v_{k'} < v_f < v_k$ and $v_{k'} < v_g < v_k$ and the right endpoint of the path that they belong, has also index less than k. Thus, $\varepsilon_{k'}^{\prime(n-1)} = \varepsilon_{k'}^{\prime(n)} + 1 = \varepsilon_{k'}^{(n)} + 1 + 1 = \varepsilon_{k'}^{(n-1)} + 1$ or $\varepsilon_{k'}^{\prime(n-1)} = \varepsilon_{k'}^{\prime(n)} + 2 = \varepsilon_{k'}^{(n)} + 1 + 2 = \varepsilon_{k'}^{(n-1)} + 2$, a contradiction. Suppose now that $t \leq k-1 \leq n$. Since there cannot exist a vertex v_ℓ such that $1 \leq \ell < t-1$ and $\varepsilon_\ell^{\prime(n)} > \varepsilon_\ell^{(n)}$, there exists a vertex $v_{k'-1}$ such that $k'-1 < k-1$ and $\varepsilon_{k'-1}^{\prime(n)} = \varepsilon_{k'-1}^{(n)}$ and $t \leq k'$. Clearly, $v_{k'} v_n \in E(G)$. If v_s is the right endpoint of P_1, then $k-1 < s$ and thus $k'-1 < s$. However, according to the algorithm, there cannot exist an endpoint between vertices v_t and v_s, a contradiction.

***Case A.3*:** When the algorithm processes vertex v_n, it constructs a 1PC of size $\lambda_{\mathcal{T}}(G[S])$ of $\mathcal{P}_{\mathcal{T}}(G[S]-v_t)$ of $G[S]-v_t$, where v_t is the terminal vertex, it connects v_t to the leftmost left endpoint it sees, and it uses v_n to bridge two paths. This operation is performed when vertex v_n sees the endpoints of at least one non-terminal path, say $P_1 = (v_r, \ldots, v_s)$, the terminal vertex v_t of the terminal path, say, $P_2 = (v_t, v_j, \ldots, v_\ell)$ and an internal vertex v_j of P_2, and it does not see v_ℓ. Then, the terminal path, P_2, has the same endpoints as in $\mathcal{P}_{\mathcal{T}}(G[S])$, the vertices of P_1 become internal vertices of P_2, while all the other paths remain the same. Recall that, when the connect operation is performed, it may use a vertex of the terminal path in order to increase the value of an $\varepsilon_i^{(n-1)}$, $1 \le i < n-1$, and, in this case, $\varepsilon_d^{(n-1)} < \varepsilon_d'^{(n-1)}$. Then, for the endpoints of the terminal path, say, $v_t \in \mathcal{T}$ and v_ℓ, we have $v_t < v_\ell$. Consequently, since vertex v_n sees the endpoints of P_1, the terminal vertex v_t and not v_ℓ, if operation connect was called previously, it cannot have used a vertex of the terminal path and, thus, $\varepsilon_d^{(n-1)} < \varepsilon_d'^{(n-1)}$ cannot hold. Consequently, $\varepsilon_i'^{(n-1)} \le \varepsilon_i^{(n-1)}$, $d \le i \le n-1$.

Consider the case where vertex v_n sees the endpoints of only one non-terminal path, that is, of P_1. If applying the algorithm to $G[S]-v_t$ results to a 1PC of size $\lambda_{\mathcal{T}}(G[S])-1$ then it contains a path with one endpoint v_k such that $t < k$. Indeed, suppose that there does not exist such a path and P_1 consists of more than one vertex. This implies that all vertices of P_1 have bridged two paths and therefore we obtain a 1PC of size less than $\lambda_{\mathcal{T}}(G[S])-1$, a contradiction. Suppose now that the algorithm does not construct a path in the 1PC of $G[S]-v_t$ with one endpoint v_k such that $t < k$ and let the path P_1 consist of one vertex, say, $v_{k'}$. This implies that vertex $v_{k'}$ has bridged two paths and the same holds for every vertex v_i, $t \le i \le n-1$. Thus, if $v_{k'}$ is removed from $G[S]-v_t$, the algorithm would construct a minimum 1PC of size $\lambda_{\mathcal{T}}(G[S])$. Since the size of $\mathcal{P}_{\mathcal{T}}(G[S])$ constructed by the algorithm is $\lambda_{\mathcal{T}}(G[S])$, removing v_t and $v_{k'}$ results to a 1PC of size $\lambda_{\mathcal{T}}(G[S])-1$. Consequently, $v_{k'}$ cannot be used for bridging two paths in $\mathcal{P}_{\mathcal{T}}(G[S]-v_t)$. This implies that the 1PC of $G[S]-v_t$ constructed by the algorithm contains a path with an endpoint v_k such that $t < k$. It is easy to see that if vertex v_n sees the endpoints of more than one non-terminal path, applying the algorithm to $G[S]-v_t$ results to a 1PC of size $\lambda_{\mathcal{T}}(G[S])-1$ having a path with one endpoint v_k such that $t < k$.

The above procedure results to a 1PC of G of size $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])-1$; consequently, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G)$.

Using similar arguments as in Case A.2, we show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i,n]$, $1 \le i \le n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i,n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \le i \le n$.

***Case A.4*:** When the algorithm processes vertex v_n, it connects v_n to a path, that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$. Suppose that there exists a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S])-1$, that is, vertex v_n is an internal vertex of a path P in $\mathcal{P}'_{\mathcal{T}}(G)$. We distinguish the following cases:

(i) $P = (v_k, \ldots, v_r, v_n, v_s, \ldots, v_\ell)$. Removing v_n from P results to a minimum 1PC of $G[S]$ having two (free) neighbors of v_n as endpoints belonging to different paths. Since the algorithm does not use v_n to bridge two paths, the constructed minimum 1PC of $G[S]$ does not have two (free) neighbors of v_n as endpoints belonging to different paths. Consequently, there is a minimum 1PC of $G[S]$ for which there exists an index i such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, a contradiction.

(ii) $P = (v_t, v_n, \ldots, v_b)$, where v_t is the terminal vertex. Removing v_n and v_t from P results to a minimum 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])-1$, a contradiction. Indeed, since the algorithm does not use v_n to bridge two paths, removing v_t from $G[S]$ results to $\lambda_{\mathcal{T}}(G[S])$ paths.

Consequently, there does not exist a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S])-1$, and, thus, the 1PC constructed by the algorithm is minimum.

Case A.4.a: Suppose that $\varepsilon_i'^{(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_T(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i,n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_T(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i,n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i < \varrho$. Furthermore, one of the following holds:

(i) $\varepsilon_i'^{(n)} \leq \varepsilon_i^{(n)}$, $\varrho \leq i \leq n$, or

(ii) if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$ then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.

Suppose that v_n is not an endpoint in $\mathcal{P}_T(G)$; let $P = (\ldots, v_a, v_n, v_b, \ldots)$. According to operation connect, we break the terminal path of $\mathcal{P}_T(G[S])$, which has the terminal vertex, v_t, as its right endpoint. Note that the terminal vertex is the second rightmost endpoint in $\mathcal{P}_T(G[S])$ (see Section 3). The second rightmost endpoint of $\mathcal{P}_T(G)$ has index greater than t, and v_t becomes a left endpoint of a path in $\mathcal{P}_T(G)$. Assume that there exists a minimum 1PC $\mathcal{P}'_T(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1,n]$, where $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$. Similarly, to Case A.1, there exists a vertex v_j, $1 \leq j < k-1$, such that $\varepsilon_j^{(n)} = \varepsilon_j'^{(n)}$.

Suppose that v_n is an endpoint of a path $P' = (v_n, v_{a'}, \ldots) \in \mathcal{P}'_T(G)$. Then, since $\varrho' = n$, $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i \leq n$. Note that, there cannot exist a minimum 1PC $\mathcal{P}'_T(G)$ such that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 2$, $\varrho \leq i \leq n$. Indeed, suppose that there exists a minimum 1PC $\mathcal{P}'_T(G)$ such that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 2$, $\varrho \leq i \leq n$. If we remove v_n from $\mathcal{P}'_T(G)$ we obtain $\varepsilon_\varrho'^{(n-1)} = \varepsilon_\varrho'^{(n)}$ or $\varepsilon_\varrho'^{(n-1)} = \varepsilon_\varrho'^{(n)} - 1$. However, $\varepsilon_\varrho'^{(n)} = \varepsilon_\varrho^{(n)} + 2$ and $\varepsilon_\varrho^{(n)} = \varepsilon_\varrho^{(n-1)} = 0$, a contradiction. According to the connect operation, $v_n v_{j+1} \notin E(G)$, thus $v_{a'} > v_{j+1}$. If we remove v_n from $\mathcal{P}'_T(G)$ we obtain $\varepsilon_{j+1}'^{(n-1)} = \varepsilon_{j+1}'^{(n)} = \varepsilon_{j+1}^{(n)} + 1$. However, $\varepsilon_{j+1}^{(n)} = \varepsilon_{j+1}^{(n-1)}$ or $\varepsilon_{j+1}^{(n)} = \varepsilon_{j+1}^{(n-1)} + 1$, a contradiction. Consequently, there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$. Suppose that $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $1 \leq i < \varrho$. Let v_r be the new endpoint created by the connect operation. Again, since $v_n v_{r-1} \notin E(G)$, $v_{a'} > v_{r-1}$ and if we remove v_n from $\mathcal{P}'_T(G)$ we obtain $\varepsilon_{r-1}'^{(n-1)} = \varepsilon_{r-1}'^{(n)} = \varepsilon_{r-1}^{(n)}$. However, $\varepsilon_{r-1}^{(n)} = \varepsilon_{r-1}^{(n-1)} + 1$, a contradiction. Consequently, if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$ then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$. Suppose that the second rightmost endpoint in $\mathcal{P}_T(G)$, say, v_f, has index less than the second rightmost endpoint in $\mathcal{P}'_T(G)$, say, $v_{f'}$, that is, $v_f < v_{f'}$. Then, $\varepsilon_{f-1}^{(n-1)} = \varepsilon_{f-1}^{(n)} - 1 = \varepsilon_{f-1}'^{(n)} - 2$ and $\varepsilon_{f-1}'^{(n-1)} = \varepsilon_{f-1}'^{(n)} - 1$, a contradiction.

Suppose that v_n is not an endpoint in $\mathcal{P}'_T(G)$; let $P' = (\ldots, v_{a'}, v_n, v_{b'}, \ldots)$. Clearly, one of $v_{a'}, v_{b'}$ is a vertex that could not be an endpoint in $\mathcal{P}'_T(G)$. We show that $\varrho' \leq \varrho$. Suppose that $\varrho' > \varrho$. Since $\varepsilon_\varrho^{(n-1)} = \varepsilon_\varrho^{(n)} = \varepsilon_\varrho'^{(n)} - 1 = 0$, then we have $\varepsilon_\varrho'^{(n-1)} = \varepsilon_\varrho'^{(n)} - 1$, which implies that the new endpoint in $\mathcal{P}'_T(G)$ has index greater than the new endpoint created in $\mathcal{P}_T(G)$, a contradiction. It is easy to see that there cannot exist a a minimum 1PC $\mathcal{P}'_T(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1,n]$ such that $\varepsilon_{k-1}'^{(n)} = \varepsilon_{k-1}'^{(n-1)} + 2$. Let v_t be the terminal vertex. We have $\varepsilon_{t-1}^{(n)} = \varepsilon_{t-1}'^{(n-1)}$. It is easy to see that there cannot exist a minimum 1PC $\mathcal{P}'_T(G)$ having an index, say, $k-1$, $k-1 \leq t-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1,n]$, such that $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$.

Suppose that v_n is the right endpoint of a path $P = (v_n, v_a, \ldots) \in \mathcal{P}_T(G)$. Then, there cannot exist a minimum 1PC $\mathcal{P}'_T(G)$ such that $1 = \varepsilon_{\varrho-1}^{(n)} < \varepsilon_{\varrho-1}'^{(n)}$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_T(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i,n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_T(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i,n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i \leq n$. Assume that there exists a minimum 1PC $\mathcal{P}'_T(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon'^{(n)}_{k-1} > \varepsilon^{(n)}_{k-1}$. Similarly, to Case 1, there exists a vertex v_j, $1 \leq j < k-1$, such that $\varepsilon^{(n)}_j = \varepsilon'^{(n)}_j$.

Suppose that v_n is an endpoint of a path $P' = (v_n, v_{a'}, \ldots) \in \mathcal{P}'_{\mathcal{T}}(G)$. Note that if $v_{a'} < v_{j+1}$, then $v_{j+1}v_n \in E(G)$ and v_k should be the terminal vertex belonging to a non-trivial path, otherwise $\mathcal{P}_{\mathcal{T}}(G)$ would not be minimum. Thus, if v_{j+1} is not the terminal vertex, $v_a < v_{j+1}$ or $v_a = v_{j+1}$. Then, if we remove v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ and $\mathcal{P}_{\mathcal{T}}(G)$, we obtain $\varepsilon^{(n-1)}_{j+1} = \varepsilon^{(n)}_{j+1} - 1 = \varepsilon'^{(n)}_{j+1} - 2$ and $\varepsilon'^{(n-1)}_{j+1} = \varepsilon'^{(n)}_{j+1} - 1$; thus, $\varepsilon^{(n-1)}_{j+1} = \varepsilon'^{(n-1)}_{j+1} - 1$, a contradiction. On the other hand, if $v_{a'} > v_{j+1}$, then, if we remove v_n from $\mathcal{P}_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon^{(n-1)}_{j+1} = \varepsilon^{(n)}_{j+1} - 1 = \varepsilon'^{(n)}_{j+1} - 2$ or $\varepsilon^{(n-1)}_{j+1} = \varepsilon^{(n)}_{j+1} = \varepsilon'^{(n)}_{j+1} - 1$. Also, $\varepsilon'^{(n-1)}_{j+1} = \varepsilon'^{(n)}_{j+1}$; thus, $\varepsilon^{(n-1)}_{j+1} = \varepsilon'^{(n-1)}_{j+1} - 1$, a contradiction.

Suppose that v_n is not an endpoint in $\mathcal{P}'_{\mathcal{T}}(G)$; let $P' = (\ldots, v_{a'}, v_n, v_{b'}, \ldots)$. If $v_{a'}v_{b'} \in E(G)$ then if we remove v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon^{(n-1)}_{j+1} = \varepsilon^{(n)}_{j+1} - 1 = \varepsilon'^{(n)}_{j+1} - 2$ or $\varepsilon^{(n-1)}_{j+1} = \varepsilon^{(n)}_{j+1} = \varepsilon'^{(n)}_{j+1} - 1$. Also, $\varepsilon'^{(n-1)}_{j+1} = \varepsilon'^{(n)}_{j+1}$; thus, $\varepsilon^{(n-1)}_{j+1} = \varepsilon'^{(n-1)}_{j+1} - 1$, a contradiction. Consequently, $v_{a'}v_{b'} \notin E(G)$; however, we have shown that v_n becomes an endpoint in $\mathcal{P}_{\mathcal{T}}(G)$ only when a new right endpoint cannot be created by making v_n an internal vertex, a contradiction.

Case A.4.b: Suppose that $\varepsilon'^{(n-1)}_i = \varepsilon^{(n-1)}_i + 1$, $d \leq i < d'$ and $\varepsilon'^{(n-1)}_i = \varepsilon^{(n-1)}_i$, $d' \leq i \leq n-1$ and there exists a vertex v_q such that $\varepsilon^{(n-1)}_q > \varepsilon'^{(n-1)}_q$, $1 \leq q < d$ and there exists no vertex $v_{q'}$, $1 \leq q' < d$, such that $\varepsilon'^{(n-1)}_{q'} > \varepsilon^{(n-1)}_{q'}$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon^{(n)}_i$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon'^{(n)}_i$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon'^{(n)}_i > \varepsilon^{(n)}_i$, $1 \leq i \leq n$.

Assume that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon'^{(n)}_{k-1}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon'^{(n)}_{k-1} > \varepsilon^{(n)}_{k-1}$. Similarly, to Case A.1, there exists a vertex v_j, $1 \leq j < k-1$, such that $\varepsilon^{(n)}_j = \varepsilon'^{(n)}_j$. Using similar arguments as in Case A.4.a, we show that v_n is an endpoint of a path $P' = (v_n, v_{a'}, \ldots) \in \mathcal{P}'_{\mathcal{T}}(G)$ and that there cannot exist a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon'^{(n)}_{k-1}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon'^{(n)}_{k-1} > \varepsilon^{(n)}_{k-1}$.

***Case A.5*:** When the algorithm processes vertex v_n, it inserts v_n into a path, that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$. This implies that $\forall i \geq d$ we have $\varepsilon^{(n-1)}_i \leq 1$. Suppose that there exists a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, that is, vertex v_n is an internal vertex of a path P in $\mathcal{P}'_{\mathcal{T}}(G)$. Then, removing vertex v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, minimum, such that there exists an index i, $i \geq d$, for which $\varepsilon'^{(n-1)}_i = 2$, a contradiction.

Consequently, there does not exist a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, and, thus, the 1PC constructed by the algorithm is minimum.

Case A.5.a: Suppose that $\varepsilon'^{(n-1)}_i \leq \varepsilon^{(n-1)}_i$, $d \leq i \leq n-1$. We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon^{(n)}_i$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon'^{(n)}_i$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon'^{(n)}_i > \varepsilon^{(n)}_i$, $1 \leq i \leq n$.

Assume that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon'^{(n)}_{k-1}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon'^{(n)}_{k-1} > \varepsilon^{(n)}_{k-1}$. Similarly, to Case A.1, there exists a vertex v_j, $1 \leq j < k-1$, such that $\varepsilon^{(n)}_j = \varepsilon'^{(n)}_j$.

Suppose that v_n is an endpoint of a path $P' = (v_n, v_t) \in \mathcal{P}'_{\mathcal{T}}(G)$, such that $v_t \in \mathcal{T}$. Then, $v_tv_n \in E(G)$ and the size of a minimum 1PC of $G[S] - v_t$ is $\lambda_{\mathcal{T}}(G) - 1$, a contradiction.

Suppose that v_n is an endpoint of a path $P' = (v_n, v_{a'}, \ldots, v_{b'}) \in \mathcal{P}'_{\mathcal{T}}(G)$, such that $v_{a'} \notin \mathcal{T}$. Then, removing vertex v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, minimum, such that there exists an index i for which $\varepsilon_i'^{(n-1)} = 1$, $i \geq d$. Then, $d = d'$, which is equal to the index of the terminal vertex, and P' is the terminal path such that its left endpoint in $\mathcal{P}'_{\mathcal{T}}(G[S])$, that is, vertex $v_{a'}$, has index greater than the index of the left endpoint of the terminal path in $\mathcal{P}_{\mathcal{T}}(G[S])$. Note that, $v_{a'}$ cannot be an endpoint in $\mathcal{P}'_{\mathcal{T}}(G[S])$ since $\varepsilon_i'^{(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$, and, thus, a 1PC of $G[S]$ having $v_{a'}$ as an endpoint cannot be minimum. However, removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$ having $v_{a'}$ as an endpoint, a contradiction.

Suppose now that v_n is not an endpoint in $\mathcal{P}'_{\mathcal{T}}(G)$; let $P' = (\ldots, v_{a'}, v_n, v_{b'}, \ldots)$. If $v_{a'}v_{b'} \in E(G)$ then if we remove v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon_{k-1}^{(n-1)} = \varepsilon_{k-1}^{(n)} = \varepsilon_{k-1}'^{(n)} - 1$ and $\varepsilon_{k-1}'^{(n-1)} = \varepsilon_{k-1}'^{(n)}$, a contradiction. Consequently, $v_{a'}v_{b'} \notin E(G)$. Suppose that the value of d-connectivity of $\mathcal{P}_{\mathcal{T}}(G[S])$ is c; then the value of d-connectivity of $\mathcal{P}_{\mathcal{T}}(G)$ is $c+2$. However, the corresponding value of $\mathcal{P}'_{\mathcal{T}}(G)$ is not increased by vertices $v_{a'}, v_n$ and $v_{b'}$, since $v_{a'}$ and $v_{b'}$ are internal vertices not successive into a path in a 1PC of $G[S]$ and there exist two vertices connected to $v_{a'}$ and $v_{b'}$ in $\mathcal{P}_{\mathcal{T}}(G[S])$, say, v_f and v_g, respectively, for which $d(v_f)$ and $d(v_g)$ are reduced, and, thus, they reduce the d-connectivity by two. In order to obtain $c+2$ for $\mathcal{P}'_{\mathcal{T}}(G)$ the vertices of $V(G) - \{v_{a'}, v_{b'}, v_n\}$ must increase the d-connectivity by two. However, the size of $\mathcal{P}'_{\mathcal{T}}(G)$ is also $\lambda_{\mathcal{T}}(G)$ and vertices $v_{a'}, v_{b'}$ and v_n are also internal in $\mathcal{P}_{\mathcal{T}}(G)$. Thus, increasing the d-connectivity by two is not possible and we have a contradiction. Note that $v_f v_g \notin E(G)$; otherwise we would have also $v_n v_f \in E(G)$ and $v_n v_g \in E(G)$ and there would exist a 1PC having the same endpoints as $\mathcal{P}'_{\mathcal{T}}(G)$ and containing a path $P = (\ldots, v_f, v_n, v_g, \ldots)$ with $v_f v_g \in E(G)$, a contradiction.

Case A.5.b: Suppose that $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)} + 1$, $d \leq i < d'$ and $\varepsilon_i'^{(n-1)} = \varepsilon_i^{(n-1)}$, $d' \leq i \leq n-1$ and there exists a vertex v_q such that $\varepsilon_q^{(n-1)} > \varepsilon_q'^{(n-1)}$, $1 \leq q < d$ and there exists no vertex $v_{q'}$, $1 \leq q' < d$, such that $\varepsilon_{q'}'^{(n-1)} > \varepsilon_{q'}^{(n-1)}$. Using similar arguments as in Case A.4.a where v_n is not an endpoint in $\mathcal{P}_{\mathcal{T}}(G[S])$, we show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i < \varrho$. Furthermore, if $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)} + 1$, $\varrho \leq i < \varrho'$ and $\varepsilon_i'^{(n)} = \varepsilon_i^{(n)}$, $\varrho' \leq i \leq n$ then there exists a vertex v_z such that $\varepsilon_z^{(n)} > \varepsilon_z'^{(n)}$, $1 \leq z < \varrho$ and there exists no vertex $v_{z'}$, $1 \leq z' < \varrho$, such that $\varepsilon_{z'}'^{(n)} > \varepsilon_{z'}^{(n)}$.

***Case A.6*:** When the algorithm processes vertex v_n, it creates a new path having vertex v_n as an endpoint, that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$. This implies that $\forall i \geq d$ we have $\varepsilon_i^{(n-1)} \leq 1$. Suppose that there exists a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, that is, vertex v_n is an internal vertex of a path P in $\mathcal{P}'_{\mathcal{T}}(G)$. Then, removing vertex v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, minimum, such that there exists an index i for which $\varepsilon_i'^{(n-1)} = 2$, a contradiction.

Suppose now that there exists a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S])$. Let v_n be an endpoint of a path P in $\mathcal{P}'_{\mathcal{T}}(G)$. We distinguish the following cases:

(i) $P = (v_n, v_r, \ldots, v_s)$. Removing vertex v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, minimum, such that there exists an index i for which $\varepsilon_i'^{(n-1)} = 1$, $i \geq d$. Then, $d = d'$, which is equal to the index of the terminal vertex, and P is the terminal path such that its left endpoint in $\mathcal{P}'_{\mathcal{T}}(G[S])$, that is, vertex v_r, has index greater than the index of the left endpoint of the terminal path in $\mathcal{P}_{\mathcal{T}}(G[S])$. Note that, v_r cannot be an endpoint in $\mathcal{P}'_{\mathcal{T}}(G[S])$ since $\varepsilon_i'^{(n-1)} \leq \varepsilon_i^{(n-1)}$, $d \leq i \leq n-1$, and, thus, a 1PC of $G[S]$ having v_r as an endpoint cannot be minimum. However, removing v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$ having v_r as an endpoint, a contradiction.

(ii) $P = (v_t, v_n)$, where v_t is the terminal vertex. Removing v_n and v_t from P results to a 1PC of $G[S] - v_t$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction. Indeed, since the algorithm does not use v_n to bridge two paths, removing v_t from $G[S]$ results to $\lambda_{\mathcal{T}}(G[S])$ paths.

Now let v_n be an internal vertex of a path $P = (v_r, \ldots, v_i, v_n, v_j, \ldots, v_s)$ in $\mathcal{P}'_{\mathcal{T}}(G)$. Suppose that $N(v_n) > 0$ (the case where $N(v_n) = 0$ is trivial) and $v_t \notin N(v_n)$, where v_t is the terminal vertex. Since the algorithm constructs $\lambda_{\mathcal{T}}(G[S]) + 1$ paths, at least $|N(v_n)| - 1$ neighbors of v_n have bridged paths reducing the size of the 1PC and at most one of them was inserted; otherwise there would exist at least two successive neighbors into a path or at least one of them would be an endpoint. Suppose that v_i and v_j have both bridged paths. Then, applying the algorithm to $G - \{v_i, v_j, v_n\}$ would result to a minimum 1PC of $G - \{v_i, v_j, v_n\}$ of size $\lambda_{\mathcal{T}}(G[S]) + 2$. However, if we remove vertices v_i, v_j and v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ we obtain a 1PC of $G - \{v_i, v_j, v_n\}$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$, a contradiction. Suppose now that v_i was inserted and v_j has bridged paths. We distinguish the following cases:

(i) $j < i$. Clearly, applying the algorithm to $G' = G - \{v_i, v_n\}$ results to a minimum 1PC of G' of size $\lambda_{\mathcal{T}}(G[S])$. Furthermore, applying the algorithm to $G' - \{v_j\}$ results to a minimum 1PC $\mathcal{P}''_1(G)$ of $G' - \{v_j\}$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$ such that no free neighbor of v_i is an endpoint and if v_i sees the terminal vertex v_t, it is not a trivial path in $\mathcal{P}''_1(G)$. Indeed, any neighbor v_a of v_i such that $i < a < n$ cannot be an endpoint in $\mathcal{P}''_1(G)$ since every vertex v_a such that $i < a < n$ is also a neighbor of v_n. Note that, $t < j$. Furthermore, since v_i is inserted, when vertex v_{j+1} was processed, no neighbor of v_i was an endpoint and if $v_i v_t \in E(G)$ vertex v_t does not belong to a trivial path. Indeed, let $v_k \in N(v_i)$ be an endpoint when the algorithm processes vertex v_{j+1} or v_t belongs to a trivial path. This implies that, when we apply the algorithm to $G[S]$, we have one neighbor of v_n, say, v_ℓ, bridging through vertex v_k or vertex v_t; then v_i would be inserted through the edge $v_k v_\ell$ or $v_t v_\ell$, which is a contradiction since this results to two neighbors of v_n being successive. Additionally, no neighbor of v_i becomes an endpoint and vertex v_t does not belong to a trivial path until vertex v_{n-1} is processed, since all vertices with index greater than $j + 1$ are neighbors of v_n, and, thus, they are used to bridge paths reducing the size of the 1PC. Note that, according to the algorithm, vertex v_t cannot belong to a trivial path until vertex v_{n-1} is processed, since no bridge operation results to v_t belonging to a trivial path. Consequently, applying the algorithm to $G' - \{v_j\}$ results to a minimum 1PC $\mathcal{P}''_1(G' - \{v_j\})$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$ such that no free neighbor of v_i is an endpoint and if v_i sees the terminal vertex v_t, it is not a trivial path in $\mathcal{P}''_1(G' - \{v_j\})$.

(ii) $i < j$. Similarly to case (i), applying the algorithm to $G' - \{v_j\}$ results to a minimum 1PC $\mathcal{P}''_1(G' - \{v_j\})$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$ such that no free neighbor of v_i is an endpoint and if v_i sees the terminal vertex v_t, it is not a trivial path in $\mathcal{P}''_1(G' - \{v_j\})$. Indeed, when vertex v_{i+1} is processed, no neighbor of v_i is an endpoint and if v_i sees the terminal vertex v_t, it is not a trivial path. Furthermore, since no neighbor of v_n can be an endpoint, no vertex with index greater than $i + 1$ is an endpoint. Additionally, no neighbor of v_i becomes an endpoint and if $v_t v_i \in E(G)$, vertex v_t does not belong to a trivial path until vertex v_{n-1} is processed, since all vertices with index greater than $i + 1$ are neighbors of v_n, and, thus, they are used to bridge paths reducing the size of the 1PC. Note that, according to the algorithm, vertex v_t cannot belong to a trivial path until vertex v_{n-1} is processed. Consequently, applying the algorithm to $G' - \{v_j\}$ results to a minimum 1PC $\mathcal{P}''_1(G' - \{v_j\})$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$ such that no free neighbor of v_i is an endpoint and if v_i sees the terminal vertex v_t, it is not a trivial path in $\mathcal{P}''_1(G' - \{v_j\})$.

Since v_n is an internal vertex of a path $P = (v_r, \dots, v_i, v_n, v_j, \dots, v_s)$ in $\mathcal{P}'_{\mathcal{T}}(G)$ which has size $\lambda_{\mathcal{T}}(G[S])$, if we remove vertices v_i, v_j and v_n from P we obtain a 1PC of $G' - \{v_j\}$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$ such that a free neighbor of v_i is an endpoint, a contradiction; the same holds when $P = (v_r, \dots, v_i, v_n, v_j, v_t)$. If $P = (v_t, v_i, v_n, v_j, \dots, v_s)$ then v_t belongs to a trivial path in $\mathcal{P}''_1(G)$, a contradiction. If $P = (v_i, v_n, v_j, \dots, v_s)$ or $P = (v_r, \dots, v_i, v_n, v_j)$, then removing vertices v_i, v_j and v_n from P results to a 1PC of $G' - \{v_j\}$ of size $\lambda_{\mathcal{T}}(G[S])$, a contradiction.

Now let $v_t \in N(v_n)$, where v_t is the terminal vertex. The case where v_n is an internal vertex of a path $P = (v_r, \dots, v_i, v_n, v_j, \dots, v_s)$ in $\mathcal{P}'_{\mathcal{T}}(G)$ which has size $\lambda_{\mathcal{T}}(G[S])$ leads to a contradiction similarly to the case where $v_t \notin N(v_n)$. Suppose that v_n is an internal vertex of a path $P = (v_t, v_n, v_j, \dots, v_s)$ in $\mathcal{P}'_{\mathcal{T}}(G)$ which has size $\lambda_{\mathcal{T}}(G[S])$. According to the algorithm, no neighbor of v_n is inserted until vertex v_t is processed. Also, it is easy to see that, no neighbor of v_n with index greater than t is inserted, either. Indeed, let v_a, $t < a < n$, be a neighbor of v_n which is inserted into the terminal path. Since the algorithm results to $\lambda_{\mathcal{T}}(G[S]) + 1$ paths, v_a is inserted through an edge $v_k v_\ell$ such that $v_k, v_\ell \notin N(v_n)$ and v_t is connected to a vertex v_q such that $v_q \notin N(v_n)$. This implies that the ordering of the vertices v_t, v_k, v_ℓ and v_q of the terminal path is as follows: $v_q < v_k < v_\ell < v_t$ or $v_q < v_\ell < v_k < v_t$; Without loss of generality suppose that $v_q < v_k < v_\ell < v_t$. Let v_b be the other endpoint of the terminal path. Clearly $v_b < v_k$. Also, without loss of generality, suppose that $v_b < v_q$. Consequently, when vertex v_t is processed, the algorithm has constructed a path having two successive vertices, v_k and v_ℓ, which have indexes greater than those of the endpoints of the path, that is, v_b and v_q. This is a contradiction, since it implies that there exists at least one vertex with index greater than q which sees v_q; in this case the algorithm could not result to a path having v_q as an endpoint. Consequently, we have shown that if we apply the algorithm to $G[S]$, no neighbor of v_n is inserted into the terminal path. Furthermore, since there are no neighbors of v_n successive into a path, all neighbors of v_n bridge paths reducing the size of the 1PC. This implies that, if we apply the algorithm to $G[S] - \{v_t\}$, we obtain a minimum 1PC of $G[S] - \{v_t\}$ of size $\lambda_{\mathcal{T}}(G[S])$. Furthermore, if we apply the algorithm to $G[S] - \{v_t, v_j\}$, we obtain a minimum 1PC of $G[S] - \{v_t, v_j\}$ of size $\lambda_{\mathcal{T}}(G[S]) + 1$. However, removing v_t, v_n and v_j from $P = (v_t, v_n, v_j, \ldots, v_s)$ which is a path in $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain a 1PC of $G[S] - \{v_t, v_j\}$ of size at most $\lambda_{\mathcal{T}}(G[S])$, a contradiction; thus, v_n cannot be an internal vertex of a path $P = (v_t, v_n, v_j, \ldots, v_s)$ in $\mathcal{P}'_{\mathcal{T}}(G)$ which has size $\lambda_{\mathcal{T}}(G[S])$.

We have shown that there does not exist a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$.

Using similar arguments as in Case A.4.a where v_n is an endpoint in $\mathcal{P}_{\mathcal{T}}(G[S])$, we show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of every interval graph G with n vertices having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \le i \le n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \le i \le n$.

Case B: vertex v_n is the terminal vertex. Clearly, the size $\lambda'_{\mathcal{T}}(G)$ of a minimum 1PC of G is equal to $\lambda_{\mathcal{T}}(G[S])$ or $\lambda_{\mathcal{T}}(G[S]) + 1$. We distinguish the following cases:

***Case B.1*:** When the algorithm processes vertex v_n, it connects v_n to a path, that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$. Since v_n is the terminal vertex, the 1PC $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S])$.

We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \le i \le n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \le i \le n$.

Suppose that $v_n \in P = (v_n, v_a, \ldots) \in \mathcal{P}_{\mathcal{T}}(G)$. Then, there cannot exist a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $1 = \varepsilon_{\varrho-1}^{(n)} < \varepsilon_{\varrho-1}'^{(n)}$. Assume that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$. Similarly, to Case A.1, there exists a vertex v_j, $1 \le j < k-1$, such that $\varepsilon_j^{(n)} = \varepsilon_j'^{(n)}$.

Suppose that $v_n \in P' = (v_n, v_{a'}, \ldots) \in \mathcal{P}'_{\mathcal{T}}(G)$. If $v_{a'} < v_{j+1}$, then $v_{j+1}v_n \in E(G)$, and, thus, $v_a < v_{j+1}$. Then, if we remove v_n from $\mathcal{P}_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}^{(n)} - 1 = \varepsilon_{j+1}'^{(n)} - 2$ and $\varepsilon_{j+1}'^{(n-1)} = \varepsilon_{j+1}'^{(n)} - 1$; thus, $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}'^{(n-1)} - 1$, a contradiction. On the other hand, if $v_{a'} > v_{j+1}$, then, if we remove v_n from $\mathcal{P}_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}^{(n)} - 1 = \varepsilon_{j+1}'^{(n)} - 2$ or $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}^{(n)} = \varepsilon_{j+1}'^{(n)} - 1$. Also, $\varepsilon_{j+1}'^{(n-1)} = \varepsilon_{j+1}'^{(n)}$; thus, $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}'^{(n-1)} - 1$, a contradiction.

***Case B.2*:** When the algorithm processes vertex v_n, it constructs a new trivial path, that is, $\lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$. Suppose that there exists a 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ of size $\lambda_{\mathcal{T}}(G[S])$. Clearly, vertex v_n cannot belong to a trivial path in $\mathcal{P}'_{\mathcal{T}}(G)$, since removing it results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S]) - 1$, a contradiction. Thus, let $P = (v_n, v_r, \ldots) \in \mathcal{P}'_{\mathcal{T}}(G)$ be the path containing v_n. Removing vertex v_n from $\mathcal{P}'_{\mathcal{T}}(G)$ results to a 1PC of $G[S]$ of size $\lambda_{\mathcal{T}}(G[S])$, and, thus, minimum, having a neighbor of v_n, that is, vertex v_r, as an endpoint of a path. Since $G[S]$ does not contain the terminal vertex this is a contradiction. Consequently, the 1PC $\mathcal{P}_{\mathcal{T}}(G)$ is a minimum 1PC of G, that is, $\lambda'_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G) = \lambda_{\mathcal{T}}(G[S]) + 1$.

We show that the algorithm computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of the graph G having $\varepsilon_i^{(n)}$ endpoints v_κ belonging to different paths with index $\kappa \in (i, n]$, $1 \leq i \leq n$, such that there is no other minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having $\varepsilon_i'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (i, n]$ such that $\varepsilon_i'^{(n)} > \varepsilon_i^{(n)}$, $1 \leq i \leq n$.

Since $P = (v_n)$ there cannot exist a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ such that $1 = \varepsilon_{\varrho-1}^{(n)} < \varepsilon_{\varrho-1}'^{(n)}$. Assume that there exists a minimum 1PC $\mathcal{P}'_{\mathcal{T}}(G)$ having an index, say, $k-1$, for which we have $\varepsilon_{k-1}'^{(n)}$ endpoints $v_{\kappa'}$ belonging to different paths with index $\kappa' \in (k-1, n]$, where $\varepsilon_{k-1}'^{(n)} > \varepsilon_{k-1}^{(n)}$. Suppose that $\varepsilon_{k-1}^{(n)} = x$ and $\varepsilon_{k-1}'^{(n)} = x + 1$. Similarly, to Case A.1, there exists a vertex v_j, $1 \leq j < k-1$, such that $\varepsilon_j^{(n)} = \varepsilon_j'^{(n)}$.

Suppose that $v_n \in P' = (v_n, v_{a'}, \ldots) \in \mathcal{P}'_{\mathcal{T}}(G)$; the case where $P' = (v_n)$ is trivial. If $v_{a'} < v_{j+1}$, then $v_{j+1}v_n \in E(G)$, and, thus, vertex v_n would be connected, a contradiction. If $v_{a'} > v_{j+1}$, then, if we remove v_n from $\mathcal{P}_{\mathcal{T}}(G)$ and $\mathcal{P}'_{\mathcal{T}}(G)$, we obtain $\varepsilon_{j+1}^{(n-1)} = \varepsilon_{j+1}^{(n)} - 1 = \varepsilon_{j+1}'^{(n)} - 2$ and $\varepsilon_{j+1}'^{(n-1)} = \varepsilon_{j+1}'^{(n)}$, a contradiction. ∎

Let $G = (V, E)$ be an interval graph on n vertices and m edges and let $\mathcal{T}$ be a terminal set containing a vertex $v \in V(G)$. Then, Algorithm Minimum_1PC computes a minimum 1PC $\mathcal{P}_{\mathcal{T}}(G)$ of G in $O(n^2)$ time and requires linear space. Recall that the ordering π of the vertices is constructed in linear time [78]. Hence, we can state the following result.

Theorem 10.2. *Let G be an interval graph on n vertices and let $\mathcal{T}$ be a subset of $V(G)$ containing a single vertex. A minimum 1-fixed-endpoint path cover of G with respect to $\mathcal{T}$ can be computed in $O(n^2)$ time.*

10.5 Related Results on Convex and Biconvex Graphs

Based on the results for the 1PC problem on interval graphs, and also on the reduction described by Müller in [69], we study the HP and 1HP problems on convex and biconvex graphs. A bipartite graph $G = (X, Y; E)$ is called *X-convex* (or simply convex) if there exists an ordering $<$ so that for all $y \in Y$ the set N(y) is $<$-consecutive [69]; G is *biconvex* if it is convex on both X and Y.

In this section, we solve the HP and 1HP problems on a biconvex graph $G = (X, Y; E)$. Moreover, we show that the HP problem on a X-convex graph $G(X, Y; E)$ on n vertices can be solved in $O(n^3)$ time if $|X| = |Y|$ or $|X| - |Y| = 1$ and a 1HP starting at vertex u, if there exists, can be computed in $O(n^2)$ time if ($|X| = |Y|$ and $u \in Y$) or $|X| - |Y| = 1$.

We next describe an algorithm for the HP problem on a biconvex graph $G = (X, Y; E)$. Note that the operation Algorithm_HP corresponds to the algorithm for computing a minimum path cover of an interval graph described in [4].

Algorithm HP_Biconvex

Input: a biconvex graph $G = (X, Y; E)$ on n vertices;

Output: a Hamiltonian path of G, if one exists;

```
1. if $||X| - |Y|| > 1$ then return($G$ does not have a Hamiltonian path);
2. if $|X| = |Y|$ then
     construct the interval graph $G'$: $V(G') = X \cup Y$, $E(G') = E \cup E_Y$, where $E_Y$ is as follows:
         $\{y_1 y_2 \in E_Y\}$ iff $y_1, y_2 \in Y$ and $N(y_1) \cap N(y_2) \neq \emptyset$;
     if $\exists y_j \in Y : |N(y_j)| = 1$ then
         $\mathcal{P}_T(G) =$Minimum_1PC($G'$,$y_j$);
         if $\lambda_T(G) = 1$ then return($\mathcal{P}_T(G)$);
         else return($G$ does not have a Hamiltonian path);
     else
         for $i = 1$ to $|Y|$ do
             $\mathcal{P}_T(G) =$ Minimum_1PC($G'$, $y_i$);
             if $\lambda_T(G) = 1$ then return($\mathcal{P}_T(G)$);
         end-for;
         return($G$ does not have a Hamiltonian path);
3. if $|X| - |Y| = 1$ then
     $\mathcal{P}_T(G) =$Algorithm_HP($G'$);
     if $\lambda_T(G) = 1$ then return($\mathcal{P}_T(G)$);
     else return($G$ does not have a Hamiltonian path);
4. if $|Y| - |X| = 1$ then
     construct the interval graph $G'$: $V(G') = X \cup Y$, $E(G') = E \cup E_X$, where $E_X$ is as follows:
       $\{x_1 x_2 \in E_X\}$ iff $x_1, x_2 \in X$ and $N(x_1) \cap N(x_2) \neq \emptyset$;
     $\mathcal{P}_T(G) =$Algorithm_HP($G'$);
     if $\lambda_T(G) = 1$ then return($\mathcal{P}_T(G)$);
     else return($G$ does not have a Hamiltonian path);
```

Algorithm 9: Algorithm HP_Biconvex

Observation 10.1. Uehara and Uno in [91] claim that the HP problem on a biconvex graph $G(X, Y; E)$ on n vertices can be solved in $O(n^2)$ time even if $|X| = |Y|$. Specifically, they claim that G has an HP if and only if the interval graph G' has an HP, where G' is an interval graph such that $V(G') = X \cup Y$ and $E(G') = E \cup E_Y$, where E_Y is as follows: $\{y_1 y_2 \in E_Y\}$ iff $y_1, y_2 \in Y$ and $N(y_1) \cap N(y_2) \neq \emptyset$. However, this is not true, since there exists a counterexample, which is presented at Figure 10.5. Indeed, the biconvex graph G of Figure 10.5 does not have an HP while for G' we have $P = (x_1, y_2, x_2, y_1, y_4, x_3, y_3, x_4)$. Suppose that we construct an interval graph G' such that $V(G') = X \cup Y$ and $E(G') = E \cup E_X$, where E_X is as follows: $\{x_1 x_2 \in E_X\}$ iff $x_1, x_2 \in X$ and $N(x_1) \cap N(x_2) \neq \emptyset$. Then, G' has an HP, that is, $P = (y_4, x_3, y_3, x_4, x_1, y_2, x_2, y_1)$. Thus, there exists no algorithm with time complexity $O(n^2)$ and we can state the following result.

Theorem 10.3. *The Hamiltonian path problem on a biconvex graph G on n vertices can be solved in $O(n^3)$ time.*

Similarly, we show that the Hamiltonian path problem on a X-convex graph $G(X, Y; E)$ on n vertices can be solved in $O(n^3)$ time when $|X| = |Y|$ or $|X| - |Y| = 1$. It is easy to see that if $|Y| - |X| = 1$ then

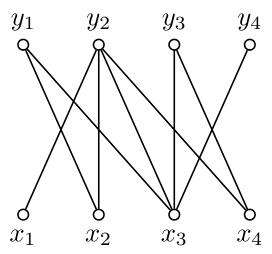


Figure 10.5: A biconvex graph G.

the X-convex graph $G(X, Y; E)$ has a Hamiltonian path if and only if the interval graph G' has a 2HP between any two vertices of Y. Thus, we can state the following result.

Corollary 10.1. *The Hamiltonian path problem on a X-convex graph $G(X, Y; E)$ on n vertices can be solved in $O(n^3)$ time if $|X| = |Y|$ or $|X| - |Y| = 1$.*

We next describe an algorithm for the 1HP problem on a biconvex graph $G = (X, Y; E)$.

Algorithm 1HP_Biconvex

Input: a biconvex graph $G = (X, Y; E)$ on n vertices and a vertex $y_t \in Y$);

Output: a Hamiltonian path of G starting at vertex y_t, if one exists;

```
1. if $||X| - |Y|| > 1$ then return($G$ does not have a 1HP);
2. if $|X| = |Y|$ then
      construct the interval graph $G'$: $V(G') = X \cup Y$, $E(G') = E \cup E_Y$, where $E_Y$ is as follows:
         $\{y_1y_2 \in E_Y\}$ iff $y_1, y_2 \in Y$ and $N(y_1) \cap N(y_2) \neq \emptyset$;
      $\mathcal{P}_\mathcal{T}(G)$ =Minimum_1PC($G', y_t$);
      if $\lambda_\mathcal{T}(G) = 1$ then return($\mathcal{P}_\mathcal{T}(G)$);
      else return($G$ does not have a 1HP);
3. if $|X| - |Y| = 1$ then return($G$ does not have a 1HP);
4. if $|Y| - |X| = 1$ then
      construct the interval graph $G'$: $V(G') = X \cup Y$, $E(G') = E \cup E_X$, where $E_X$ is as follows:
         $\{x_1x_2 \in E_X\}$ iff $x_1, x_2 \in X$ and $N(x_1) \cap N(x_2) \neq \emptyset$;
      $\mathcal{P}_\mathcal{T}(G)$ =Minimum_1PC($G', y_t$);
      if $\lambda_\mathcal{T}(G) = 1$ then return($\mathcal{P}_\mathcal{T}(G)$);
      else return($G$ does not have a 1HP);
```

Algorithm 10: Algorithm 1HP_Biconvex

Since Algorithm Minimum_1PC requires $O(n^2)$ time to compute a 1HP of an interval graph on n vertices and the graph G' can be constructed in $O(|X \cup Y|^2)$ time [69], Algorithm 1HP_Biconvex returns a 1HP, if there exists, of a biconvex graph on n vertices in $O(n^2)$ time. Hence, we can state the following result.

Theorem 10.4. *Let G be a biconvex graph on n vertices and let u be a vertex of $V(G)$. The 1HP problem on G can be solved in $O(n^2)$ time.*

Let $G(X, Y; E)$ be a X-convex graph on n vertices and let u be a vertex of $V(G)$. Similarly, we show that the 1HP problem on G can be solved in $O(n^2)$ time when ($|X| = |Y|$ and $u \in Y$) or $|X| - |Y| = 1$. Clearly, if $|Y| - |X| = 1$ and $u \in X$ then G does not have a 1HP. It is easy to see that if ($|Y| - |X| = 1$ and $u \in Y$) or ($|X| = |Y|$ and $u \in X$) then the X-convex graph $G(X, Y; E)$ has a Hamiltonian path if and only if the interval graph G' has a 2HP between u and a vertex of Y. Thus, we can state the following result.

Corollary 10.2. *Let $G(X, Y; E)$ be a X-convex graph on n vertices and let u be a vertex of $V(G)$. The 1HP problem on G can be solved in $O(n^2)$ time if ($|X| = |Y|$ and $u \in Y$) or $|X| - |Y| = 1$. If $|Y| - |X| = 1$ and $u \in X$ then G does not have a 1HP.*

10.6 Concluding Remarks

This chapter presents an $O(n^2)$ time algorithm for the 1PC problem on interval graphs. Given an interval graph G and a vertex v of G, our algorithm constructs a minimum path cover of G such that v is an endpoint. Thus, if the graph G is Hamiltonian, our algorithm constructs a 1HP. It would be interesting to see if the problem can be solved in linear time. Furthermore, an interesting open question is whether the k-fixed-endpoint path cover problem (kPC) can be polynomially solved on interval graphs. Given a graph G and a subset $\mathcal{T}$ of k vertices of $V(G)$, a k-fixed-endpoint path cover of G with respect to $\mathcal{T}$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the k vertices of $\mathcal{T}$ are all endpoints of the paths in $\mathcal{P}$. The kPC problem is to find a k-fixed-endpoint path cover of G of minimum cardinality. Note that, the kPC problem generalizes the 2HP problem; the complexity status of the 2HP problem on interval graphs remains an open question.

Chapter 11

Conclusions-Further Research

11.1 Coloring Problems

The complexity of the pair-complete coloring problem and the harmonious coloring problem has been extensively studied on various classes of perfect graphs such as cographs, interval graphs, bipartite graphs and trees. Chapter 4 of our work is also concerned with these problems.

Bodlaender [11] provides a proof for the NP-completeness of the pair-complete coloring problem for disconnected cographs and disconnected interval graphs, and extends his results for the connected cases. His proof also establishes the NP-hardness of the harmonious coloring problem for disconnected interval graphs and disconnected cographs. Bodlaender's results establish the NP-hardness of the pair-complete coloring problem for the class of permutation graphs and, also, the NP-hardness of the harmonious coloring problem when restricted to disconnected permutation graphs. Extending the above results, we have shown that the harmonious coloring problem remains NP-complete on connected interval and permutation graphs. Furthermore, we also show that the problem is also NP-complete on split graphs.

Concerning the class of bipartite graphs and subclasses of this class Farber et al. [32] show that the harmonious coloring problem and the pair-compete coloring problem are NP-complete for the class of bipartite graphs. In addition, Edwards et al. [30, 31] show that these problems are NP-complete for trees. Their results also establish the NP-completeness of these problems for the classes of convex graphs and disconnected bipartite permutation graphs. We have shown that the harmonious coloring problem and the pair-complete coloring problem is NP-complete for connected bipartite permutation graphs, and thus, the same holds for the class of biconvex graphs. Moreover, based on Bodlaender's results [11], we showed that the pair-complete coloring problem is NP-complete for quasi-threshold graphs and that the harmonious coloring problem is NP-complete for disconnected quasi-threshold graphs. We have also showed that the pair-complete coloring problem is polynomially solvable on threshold graphs by proposing a simple linear-time algorithm.

11.2 Path Cover Problems

Nakano et al. in [70] use novel techniques that combine in a clever way tree structures, called path trees, and sequences of square and round brackets; their algorithm produces a minimum path cover of a cograph by finding matchings of brackets in these sequences, constructing path trees, and converting the path trees to a minimum path cover using the inorder traversal. In [70] they left open the problem of applying their technique into other classes of graphs. We generalized their technique and applied it to the class of P_4-sparse graphs. We investigated the structure of the paths that occur in a minimum path cover of a P_4-sparse graph and the structure of the corresponding path trees, and presented a time- and work-optimal algorithm that runs in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. We also showed that our results can be extended to a proper superclass of P_4-sparse graphs, namely the P_4-tidy graphs.

Steiner [87] showed that the k-path partition problem remains NP-complete on the class of chordal bipartite graphs if k is part of the input and on the class of comparability graphs even for $k = 3$. Furthermore, he presented a polynomial time solution for the problem, with any k, on bipartite permutation graphs and left the problem open for the class of convex graphs. In Chapter 3 we proved that the k-path partition problem is NP-complete on convex graphs, quasi-threshold graphs, and thus, it is also NP-complete for interval and chordal graphs. Given that this problem is polynomially solvable for bipartite permutation graphs, we have sharpened the demarcation line between polynomially solvable and NP-hard cases of the k-path partition problem.

In Chapter 5 we presented a simple linear-time algorithm for the k-fixed-endpoint path cover problem (kPC) on cographs. Given a cograph G and a set of vertices $\mathcal{T}$, our algorithm constructs a minimum k-fixed-endpoint path cover of G that contains a large number of terminal paths. Furthermore, we studied the complexity status of the k-fixed-endpoint path cover problem on the class of proper interval graphs, and showed that this problem can be solved in polynomial time when the input is a proper interval graph.

A generalization of the (kPC) problem has been defined and also solved for the class of cographs, namely, the k-fixed-set path cover problem (kFSPC). Recall that, given a graph G and k disjoint subsets $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ of $V(G)$, a k-fixed-endpoint-set path cover of G is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are all endpoints of the paths in $\mathcal{P}$ and two vertices $u, v \in \mathcal{T}$ belong to the same path of $\mathcal{P}$ if both u, v belong to the same set $\mathcal{T}^i$, $i \in [1, k]$. The proposed algorithm produces a minimum k-fixed-set path cover of a cograph G that contains the largest number of terminal paths.

Furthermore, generalizing the disjoint paths problem, we have introduced the 2-terminal-set path cover problem (2TPC) and we have proposed a polynomial-time solution on the class of cographs. Given a graph G and two disjoint subsets $\mathcal{T}^1$ and $\mathcal{T}^2$ of $V(G)$, a 2-terminal-set path cover of G with respect to $\mathcal{T}^1$ and $\mathcal{T}^2$ is a set of vertex-disjoint paths $\mathcal{P}$ that covers the vertices of G such that the vertices of $\mathcal{T}^1$ and $\mathcal{T}^2$ are all endpoints of the paths in $\mathcal{P}$ and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2$ have one endpoint in $\mathcal{T}^1$ and the other in $\mathcal{T}^2$. Our algorithm produces a minimum 2-terminal-set path cover of a cograph G that contains the largest number of terminal paths. The $k(2)$-terminal-set path cover problem (k(2)TSPC) is presented in Chapter 8 along with a polynomial-time solution on the class of cographs. Let G be a graph and let $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ be disjoint sets of vertices of $V(G)$ each containing at most two vertices. Recall that, a $k(2)$-terminal-set path cover of the graph G with respect to $\mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^k$ is a path cover of G such that all vertices in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ are endpoints of paths in the path cover and all the paths with both endpoints in $\mathcal{T}^1 \cup \mathcal{T}^2 \cup \ldots \cup \mathcal{T}^k$ have one endpoint in $\mathcal{T}^i$ and the other in $\mathcal{T}^j$, $i \neq j$ and $i, j \in [1, k]$. Again, our algorithm produces a minimum $k(2)$-terminal-set path cover of a cograph G that contains the largest number of terminal paths.

Finally, motivated by a work of Damaschke [27], where he left both 1HP and 2HP problems open for the class of interval graphs, we have presented an $O(n^2)$ time algorithm for the 1PC problem on interval graphs. Given an interval graph G and a vertex v of G, our algorithm constructs a minimum path cover of G such that v is an endpoint. Thus, if the graph G is Hamiltonian, our algorithm constructs a 1HP.

11.3 Further Research

In Chapter 2 we presented an optimal parallel algorithm for solving the minimum path cover problem on P_4-sparse graphs. An interesting open question would be to see if similar techniques can be efficiently used for finding a minimum path cover for other classes of graphs and solving other related algorithmic problems such as the k-fixed-endpoint path cover problem.

Concerning the k-path partition problem, we have sharpened the demarcation line between polynomially solvable and NP-hard cases of the problem. However, the status of the k-path partition problem remains open for the class of biconvex graphs; this class properly contains bipartite permutation graphs and is a proper subclass of convex graphs.

It would be interesting to see if the problems introduced in this work, that is the kPC, the kFSPC, the 2TPC and the k(2)TSPC can be polynomially solved on other classes of graphs; an interesting next step would be to consider the class of interval graphs. This promises to be an interesting area for further research since the complexity status of a simpler problem, that is, the 2HP problem, is unknown for interval graphs. Note that, the complexity status of the kTSPC problem on cographs is unknown, we pose it as an open problem.

Finally, we have presented an $O(n^2)$ time algorithm for the 1PC problem on interval graphs. It would be interesting to see if the problem can be solved in linear time. Furthermore, an interesting open question is whether the k-fixed-endpoint path cover problem (kPC) can be polynomially solved on interval graphs. Note that, the kPC problem generalizes the 2HP problem; the complexity status of the 2HP problem on interval graphs remains an open question. As a first step, it is worth checking if the 1PC algorithm proposed in Chapter 10 can be extended to solve the 2PC problem on interval graphs.

Bibliography

[1] K. Abrahamson, N. Daboun, D.G. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm, *J. Algorithms* **10** (1989) 287–302.

[2] G.S. Adhar and S. Peng, Parallel algorithm for path covering, Hamiltonian path, and Hamiltonian cycle in cographs, *Int'l Conference on Parallel Processing*, Vol. III: Algorithms and Architecture, Pennsylvania State University Press, pp. 364–365, 1990.

[3] T. Akiyama, T. Nishizeki, and N. Saito, NP-completeness of the Hamiltonian cycle problem for bipartite graphs, *J. Inform. Process.* **3** (1980) 73–76.

[4] S.R. Arikati and C.P. Rangan, Linear algorithm for optimal path cover problem on interval graphs, *Inform. Process. Lett.* **35** (1990) 149–153.

[5] J. Bang-Jensen, J. Huang, and L. Ibarra, Recognizing and representing proper interval graphs in parallel using merging and sorting, *Discrete Appl. Math.* **155** (2007) 442–456.

[6] C. Berge, Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind, Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe **10** (1961) 114–115.

[7] C. Berge, *Graphs and Hypergraphs*, American Elsevier, New York, 1973.

[8] A.A. Bertossi, Finding Hamiltonian circuits in proper interval graphs, *Inform. Process. Lett.* **17** (1983) 97–101.

[9] A.A. Bertossi and M.A. Bonuccelli, Finding Hamiltonian circuits in interval graph generalizations, *Inform. Process. Lett.* **23** (1986) 195–200.

[10] N. Biggs, *Algebraic Graph Theory*, Cambridge University Press, London, 1974.

[11] H.L. Bodlaender, Achromatic number is NP-complete for cographs and interval graphs, *Inform. Process. Lett.* **31** (1989) 135–138.

[12] K.S. Booth and G.S. Lueker, Testing for the consecutive ones property, interval graphs, anf graph planarity using PQ-tree algorithms, *J. Comput. System Sci.* **13** (1976) 335–379.

[13] A. Brandstädt, V.B. Le, and J. Spinrad, *Graph classes – A survey*, SIAM Monographs in Discrete Mathematics and Applications, SIAM, Philadelphia, 1999.

[14] A. Bretscher, D. Corneil, M. Habib, and C. Paul, A simple linear time LexBFS cograph recognition algorithm, *Proc. 29th Intől Workshop on Graph Theoretic Concepts in Comput. Sci. (WGŏ03)*, LNCS 2880 (2003) 119ΰ-130.

[15] N. Cairnie and K.J. Edwards, Some results on the achromatic number, *J. Graph Theory* **26** (1997) 129–136.

[16] M. Chudnovsky, N. Robertson, P.D. Seymour, and R. Thomas, The strong perfect graph theorem, *Annnals of Math.* **164** (2006) 51–229.

[17] V. Chvátal and P.L. Hammer, Aggregation of inequalities in integer programming, *Annals of Discrete Mathematics*, 145–162, North-Holland, Amsterdam, 1977.

[18] V. Chvátal and P.L. Hammer, Set-packing and threshold graphs, *Res. Report CORR* 73–21, University of Warerloo, 1973.

[19] S. Cook, The complexity of theorem-proving procedures, *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, Association for Computing Machinery, New York, 151–158.

[20] C.J. Colburn and L.K. Stewart, Dominating cycles in series-parallel graphs, *Ars Combin.* **19A** (1985) 107–112.

[21] D.G. Corneil, H. Kim, S. Nataranjan, S. Olariu, and A.P. Sprague, Simple linear time recognition of unit interval graphs, *Inform. Process. Lett.* **55** (1995) 99–104.

[22] D.G. Corneil, H. Lerchs, and L. Stewart Burlingham, Complement reducible graphs, *Discrete Appl. Math.* **3** (1981), 163–174.

[23] D.G. Corneil and Y. Perl, Clustering and domination in perfect graphs, *Discrete Appl. Math.* **9** (1984), 27–39.

[24] D.G. Corneil, Y. Perl, and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **14** (1985) 926–984.

[25] E. Dalhaus, Efficient parallel modular decomposition, *21st Int'l Workshop on Graph Theoretic Concepts in Computer Science (WG'95)*, LNCS **1017** (1995) 290–302.

[26] E. Dalhaus, J. Gustedt and R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* **41** (2001) 360–387.

[27] P. Damaschke, Paths in interval graphs and circular arc graphs, *Discrete Math.* **112** (1993) 49–64.

[28] P. Damaschke, J.S. Deogun, D. Kratsch, and G. Steiner, Finding Hamiltonian paths in cocomparability graphs using the bump number algorithm, *Order* **8** (1992) 383–391.

[29] X. Deng, P. Hell, and J. Huang, Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs, *SIAM J. Comput.* **25** (1996) 390–403.

[30] K.J. Edwards, The harmonious chromatic number and the achromatic number, in: *Surveys in Combinatorics* (R.A. Baily, ed.), Cambridge University Press, Cambridge (1997) 13–47.

[31] K.J. Edwards and C. McDiarmid, The complexity of harmonious coloring for trees, *Discrete Appl. Math.* **57** (1995) 133–144.

[32] M. Farber, G. Hahn, P. Hell, and D. Miller, Concerning the achromatic number of graphs, *J. Combinatorial Theory*, Series B, **40** (1986) 21–39.

[33] S. Fortune, J.E. Hopcroft, and J. Wyllie, The directed subgraph homeomorphism problem, *J. Theoret. Comput. Sci.* **10** (1980) 111–121.

[34] O. Frank, F. Harary, and M. Plantholt, The line-distinguishing chromatic number of a graph, *Ars Combin.* **14** (1982) 241–252.

[35] M.R. Garey and D.S. Johnson, “Strong” NP-completeness results: Motivation, exapmles, and implications, *J. of the Association for Computing Machinery*, **25** (1978) 499–508.

[36] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, 1979.

[37] M.R. Garey, D.S. Johnson, and R.E. Tarjan, The planar Hamiltonian circuit problem is NP-complete, *SIAM J. Comput.* **5** (1976) 704–714.

[38] V. Giakoumakis, F. Roussel and H. Thuillier, On P_4-tidy graphs, *Discrete Math. and Theoret. Comput. Science* **1** (1997) 17–41.

[39] V. Giakoumakis and J-M. Vanherpe, On extended P_4-reducible and P_4-sparse graphs, *Theoret. Comput. Science* **180** (1997) 269–286.

[40] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980. Second edition, Annals of Discrete Math. **57**, Elsevier, 2004.

[41] F. Harary, *Graph Theory*, Addison-Wesley, 1969.

[42] F. Harary and S.T. Hedetniemi, The achromatic number of a graph, *J. Combin. Theory* **8** (1970) 154–161.

[43] F. Harary, S.T. Hedetniemi, and G. Prins, An interpolation theory for graphical homomorphisms, *Portugal. Math.* **26** (1967) 453–462.

[44] P. Hell, R. Shamir, and R. Sharan, A fully dynamic algorithm for recognizing and representing proper interval graphs, *SIAM J. Comput.* **31** (2001) 289–305.

[45] C. Hoàng, Perfect graphs, PhD Thesis, McGill University, Montreal, Canada, 1985.

[46] W. Hochstättler and G. Tinhofer, Hammiltonicity in graphs with few P_4's, *Computing* **54** (1995) 213–225.

[47] J.E. Hopcroft and M.S. Krishnamoorthy, On the harmonious coloring of graphs, *SIAM J. Alg. Discrete Meth.* **4** (1983) 306–311.

[48] S.Y. Hsieh, An efficient parallel strategy for the two-fixed-endpoint Hamiltonian path problem on distance-hereditary graphs, *J. Parallel Distrib. Comput.* **64** (2004) 662–685.

[49] S.Y. Hsieh, C.W. Ho, T.S. Hsu, and M.T. Ko, The Hamiltonian problem on distance-hereditary graphs, *Discrete Appl. Math.* **154** (2006) 508–524.

[50] R.W. Hung and M.S. Chang, Linear-time algorithms for the Hamiltonian problems on distance-hereditary graphs, *Theoret. Comput. Sci.* **341** (2005) 411–440.

[51] R.W. Hung and M.S. Chang, Solving the path cover problem on circular-arc graphs by using an approximation algorithm, *Discrete Appl. Math.* **154** (2006) 76–105.

[52] J. Jájá, An introduction to parallel algorithms, Addison-Wesley, Reading, MA, 1992.

[53] B. Jamison and S. Olariu, A new class of brittle graphs, *Studies Appl. Math.* **81** (1989) 89–92.

[54] H.A. Jung, On a class of posets and the corresponding comparability graphs, *J. Combinatorial Theory (B)* **24** (1978), 125–133.

[55] M. Kano and S.D. Nikolopoulos, On the Structure of A-free Graphs: Part II , TR–1999-25, Department of Computer Science, University of Ioannina, 1999.

[56] R.M. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations*, Plenum Press, New York, 85–103, 1972.

[57] R.M. Karp, On the complexity of combinatorial problems, *Networks* **5** (1975), 45–68.

[58] J.M. Keil, Finding Hamiltonian circuits in interval graphs, *Inform. Process. Lett.* **20** (1985) 201–206.

[59] H. Lerchs, On cliques and kernels, Department of Computer Science, University of Toronto, March 1971.

[60] R. Lin and S. Olariu, A fast parallel algorithm to recognize P4-sparse graphs, *Discrete Appl. Math.* **81** (1998) 191–215.

[61] R. Lin, S. Olariu and G. Pruesse, An optimal path cover algorithm for cographs, *Comput. Math. Appl.* **30** (1995) 75–83.

[62] R. Lin, S. Olariu, J.L. Schwing and J. Zhang, A fast EREW algorithm for minimum path cover and hamiltonicity for cographs, *Parallel Algorithms Appl.* **2** (1994) 99–113.

[63] W. Lipski and F.P. Preparata, Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems, *Acta Informatica* **15** (1981) 329–346.

[64] P.J. Looges and S. Olariu, Optimal greedy algorithms for indifference graphs, *Comput. Math. Appl.* **25** (1993) 15–25.

[65] L. Lovász, A characterization of perfect graphs, *J. Combinatorial Theory, Ser. B* **13** (1972) 95–98.

[66] S. Ma, W.D. Wallis, and J. Wu, Optimization problems on quasi-threshold graphs, *J. Comb. Inform. and Syst. Sciences* **14** (1989) 105–110.

[67] F. Margot, Some complexity results about threshold graphs, *Discrete Appl. Math.* **49** (1994) 299–308.

[68] R.M. McConnell and J. Spinrad, Modular decomposition and transitive orientation, *Discrete Math.* **201** (1999) 189–241.

[69] H. Müller, Hamiltonian circuits in chordal bipartite graphs, *Discrete Math.* **156** (1996) 291–298.

[70] K. Nakano, S. Olariu, and A.Y. Zomaya, A time-optimal solution for the path cover problem on cographs, *Theoret. Comput. Sci.* **290** (2003) 1541–1556.

[71] G. Narasimhan, A note on the Hamiltonian circuit problem on directed path graphs, *Inform. Process. Lett.* **32** (1989) 167–170.

[72] S.D. Nikolopoulos, Parallel algorithms for Hamiltonian problems on quasi-threshold graphs, *J. Parallel Distrib. Comput.* **64** (2004) 48–67.

[73] S.D. Nikolopoulos, Recognizing cographs and threshold graphs through a classification of their edges, *Inform. Process. Lett.* **75** (2000) 129–139.

[74] S.D. Nikolopoulos and L. Palios, Hole and antihole detection in graphs, *Proc. 15th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'04)*, (2004) 850–859.

[75] B.S. Panda and S.K. Das, A linear time recognition algorithm for proper interval graphs, *Inform. Process. Lett.* **87** (2003) 153–161.

[76] I. Parfenoff, An efficient parallel algorithm for maximum matching for some classes of graphs, *J. Parallel and Distributed Computing* **52** (1998) 96–108.

[77] J.H. Park, One-to-Many disjoint path covers in a graph with faulty elements, *Proc. 10th Internat. Computing and Combinatorics Conf. (COCOON'04)*, (2004) 392ΰ-401.

[78] G. Ramalingam and C.P. Rangan, A unified approach to domination problems on interval graphs, *Inform. Process. Lett.* **27** (1988), 271–274.

[79] J. Reif (editor), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, Inc., 1993.

[80] F.S. Roberts, *Indifference graphs*, Proof Techniques in Graph Theory, F. Harary, ed., Academic Press, New York, 1969, 139–146.

[81] F.S. Roberts, Graph theory and its applications to problems of society, SIAM Press, Philadelphia, PA, 1978.

[82] N. Robertson and P.D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Combinatorial Theory*, Series B, **63** (1995) 65–110.

[83] P.D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980) 293–309.

[84] Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. Assoc. Comput. Mach.* **27** (1980) 445–456.

[85] J. Spinrad, A. Brandstadt, and L. Stewart, Bipartite permutation graphs, *Discrete Appl. Math.* **18** (1987) 279–292.

[86] R. Srikant, R. Sundaram, K.S. Singh, and C.P. Rangan, Optimal path cover problem on block graphs and bipartite permutation graphs, *Theoret. Comput. Sci.* **115** (1993) 351–357.

[87] G. Steiner, On the k-path partition of graphs, *Theoret. Comput. Science* **290** (2003) 2147–2155.

[88] G. Steiner, On the k-path partition problem in cographs, *Cong. Numer.* **147** (2000) 89–96.

[89] Y. Suzuki, K. Kaneko, and M. Nakamori, Node-disjoint paths algorithm in a transposition graph, *IEICE Trans. Inform. & Syst.* **E89-D** (2006) 2600–2605.

[90] C. Thomassen, 2-linked graphs, *Europ. J. Combin.* **1** (1980 371–378.

[91] R. Uehara and Y. Uno, On computing longest paths in small graph classes, *Int. J. Foundations Comput. Science* **18** (2007) 911–930.

[92] J. Yan, G.J. Chang, S.M. Hedetniemi, and S.T. Hedetniemi, k-path partitions in trees, *Discrete Appl. Math.* **78** (1997) 227–233.

[93] M. Yannakakis and F. Gavril, Edge dominating sets in graphs, *SIAM J. Appl. Math.* **38** (1980) 364–372.

INDEX

Curriculum Vitae

Katerina Asdre was born in November, 1977. She received the B.Sc degree and the M.Sc degree in Computer Science from the University of Ioannina, Greece, in 1999 and 2001, respectively. Since 2003 she has been a Ph.D student in the Department of Computer Science at the University of Ioannina. Specifically:

4/2003–6/2008	Ph.D in Computer Science Department of Computer Science, University of Ioannina. Ph.D thesis: *Algorithms and NP-completeness Results for Variations of Path Cover and Coloring Problems.*
9/1999–9/2001	M.Sc in Computer Science Department of Computer Science, University of Ioannina. M.Sc thesis: *Tree-Like Data Structures for Priority Queues in Discrete Event Simulation Systems.*
10/1995–9/1999	B.Sc in Computer Science Department of Computer Science, University of Ioannina. B.Sc thesis: *Algorithms and Data Structures for Discrete Event Simulation Systems.*

Her research interests are in design and analysis of algorithms, parallel algorithms, data structures and algorithms, graph algorithms, simulation techniques, discrete event systems. She has studied sequential and parallel algorithms for optimization problems as well as NP-complete problems on classes of perfect graphs. Her research work has been published at journals and conferences.

Publications

1. K. Asdre and S.D. Nikolopoulos, NP-completeness results for some problems on subclasses of bipartite and chordal graphs, *Theoret. Comput. Science* **381**, 248–259, 2007.

2. K. Asdre, S.D. Nikolopoulos and Ch. Papadopoulos, An optimal parallel solution for the path cover problem on P_4-sparse graphs, *J. Parallel and Distributed Computing* **67**, 63–76, 2007.

3. K. Asdre and S.D. Nikolopoulos, A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs, *Networks* **50**, 231–240, 2007.

4. K. Asdre, K. Ioannidou, and S.D. Nikolopoulos, The harmonious coloring problem is NP-complete for interval and permutation graphs, *Discrete Appl. Math.* **155**, 2377–2382, 2007.

5. K. Asdre and S.D. Nikolopoulos, P-tree structures and event horizon: Efficient event-set implementations, *J. Computer Science and Technology* **21**, 19–26, 2006.

6. K. Asdre and S.D. Nikolopoulos, The 2-terminal-set path cover problem and its polynomial solution on cographs, *Frontiers of Algorithmics Workshop (FAW '08)*, Changsha, China, 2008.

7. K. Asdre and S.D. Nikolopoulos, A polynomial solution for the k-fixed-endpoint path cover problem on proper interval graphs, *Proc. 18th International Conference on Combinatorial Algorithms (IWOCA'07)*, Lake Macquarie, Newcastle, Australia, 2007.

8. K. Asdre, S.D. Nikolopoulos and Ch. Papadopoulos, Optimal algorithms for the path cover problem on P_4-sparse graphs, *Workshop on Graphs and Combinatorial Optimization (CTW'05)*, Vol. 1, 79–83, 2005.

9. K. Asdre and S.D. Nikolopoulos, P-tree structures and event horizon: Efficient event-set implementations, *Winter Simulation Conference (WSC'05)*, Orlando, Florida, 2005.