Exact and heuristic algorithms for network alignment using graph edit distance models

Thesis for obtaining the title of
Doctor of Natural Science
of the Faculty of Natural Science and Technology I of
Saarland University

by
Rashid Ibragimov, M.Sc.

Saarbrücken
August, 2014

Day of Colloquium	Feb. 5, 2015
Dean of the Faculty	Prof. Dr. Markus Bläser
Chair of the Committee	Prof. Dr. Volkhard Helms
First reviewer	Prof. Dr. Jan Baumbach
Second reviewer	Prof. Dr. Jiong Guo
Third reviewer	Prof. Dr. Raimund Seidel
Academic Assistant	Dr. Marc Hellmuth

Zusammenfassung

In dieser Arbeit werden theoretische und praktische Aspekte der Anwendung des GED(Graph Edit Distance)-Modells auf PPI (Protein-Protein-Interaktions)-Netzwerke untersucht. Hierbei werden werden ausschließlich topologische Informationen von Graphen verwendet. In zweiten Teil werden einige theoretische Eigenschaften des Modells untersucht, formuliert als drei verschiedene Problemstellungen. Im dritten Teil werden drei Heuristiken zur approximativen Lösung des PPI-Netzwerk-Alignmentproblems präsentiert, basierend auf einem GED-Modell, dass die Anzahl gelöschter und neu eingefügter Kanten auswertet.
Es folgen Zusammenfassungen der wichtigsten Kapitel dieser Arbeit.

Baumüberdeckung mit Sterngraphen (CTS). In Kapitel 3 werden die Komplexitätsgrenzen einer der einfachsten Formulierungen der GED – Baum-Editierdistanz mit Einfügen und Löschen von Kanten (Tree Edit Distance with edge Insertions and Deletions, TED-ID) untersucht. Dabei wird ein spezieller Fall betrachtet und als CTS umformuliert. CTS untersucht ob es möglich ist, gegeben ein Baum und eine Menge Sterngraphen, so Kanten zu den Sterngraphen hinzuzufügen, dass der resultierende Graph isomorph zum Baum ist. Das wichtigste Ergebnis von Kapitel 3 ist der Beweis der NP-Schwere von CTS. Aus diesem Beweis folgt, dass auch TED-ID NP-Schwer ist, selbst wenn der Durchmesser der Eingabebäume auf maximal zehn begrenzt ist. Wir beweisen außerdem, dass CTS in polynomieller Zeit lösbar ist, wenn der größte Stern nicht größer als eine Konstante k ist.

Nachbarschaftserhaltende Zuordnung (NPM) auf Bäumen. In Kapitel 4 wird NPM, eine neue Generalisierung des Graphisomorphismusproblems eingeführt. Die Eingabe von NPM umfasst zwei Graphen und drei Ganzzahlen, l, d und k. Das Problem besteht darin, den ersten Graphen ohne höchstens k Kanten (diese werden Isolationskanten genannt) auf den zweiten abzubilden. Mittels der Parameter l und d wird für jeden betrachteten Knoten v des ersten Graphen spezifiziert welche Nachbarn von v (innerhalb einer Umgebung der Größe l) auch im zweiten Graphen nah genug beim Bild von v liegen müssen (innerhalb einer

Umgebung der Größe d). Für NPM wird angenommen, dass Graphen mit einer kleineren Isolationsmenge sich ähnlicher sind, da weniger Editierungsoperationen benötigt werden um diese ineinander umzuwandeln. Das Hauptergebnis der Untersuchung von NPM auf Bäumen betrifft dessen Komplexität. Während bestimmte Werte für l, d und k eine Berechnung in polynomieller Zeit erlauben, führen andere zu einem NP-schwerem Problem. Interessanterweise ist NPM auf Bäumen sogar dann NP-schwer, wenn die Abbildung den ersten Baum in den zweiten "quetscht", das heißt für die Fälle dass $d > l$ und $l \geq 3$. Obwohl alle Beweise der NP-Schwere in dieser Arbeit auf Reduktionen des selben Problems (3-Partition) basieren, ist der Schlüssel jedes Beweises das Verständnis der Struktur der zu konstruierenden Bäume in Abhängigkeit der Eingabeparameter.

Kompaktheitserhaltende Zuordnung (CPM) auf Bäumen. In Kapitel 5 wird CPM, eine weitere Generalisierung des Graphisomorphismusproblems, betrachtet. Die Eingabe von CPM besteht aus zwei Graphen und zwei Ganzzahlen l und d, wobei l die gleiche Bedeutung hat wie in NPM. Der Parameter d entspricht dem "Gesamtfehler" der erlaubt ist wenn Nachbarn eines Knotens v im ersten Graphen auf den zweiten abgebildet werden: Gegeben die Summe L_v der Distanzen zwischen v und dessen nächsten Nachbarn, sowie die Summe L'_v der Distanzen zwischen den Bildern von v und den Bildern der nächsten Nachbarn, untersucht CPM ob es eine eins-zu-eins-Abbildung zwischen beiden Graphen gibt, sodass für jeden Knoten v gilt: $L'_v - L_v \leq d$. Einerseits beschränkt CPM die Formulierung von NPM mit $k = 0$ und lockert sie andererseits: Im Gegensatz zu NPM ist es möglich, dass einige Kanten weiter entfernt als d abgebildet werden, aber die meisten Kanten sind näher beisammen. Es wird erwartet, dass zwei Graphen ähnlicher zueinander sind je kleiner der Wert d für eine Lösung von CPM gewählt werden kann. In Kapitel 5 wird CPM auf Bäumen untersucht und eine Gegensätzlichkeit der Komplexität von CPM bezüglich verschiedener Werte von l und d päsentiert.

Netzwerkalignment mittels BCO. Für praktische Anwendung sind effiziente Heuristiken vonnöten, die eine optimale Lösung aproximieren können, insbesondere da die Berechnung der Graph-Editierdistanz (GED) NP-schwer ist. In Kapitel 8 wird die erste Heuristik dieser Arbeit, NABEECO, beschrieben. NABEECO approximiert die GED mit Hilfe einer nachgeahmten Bienenkolonie, die den Suchraum durchwandert und die Lösung dadurch verfeinert. Wir demonstrieren die Leistungsfähigkeit von NABEECO anhand echter Protein-Protein-Interaktionsdaten. Trotz guter Ergebnisse ist die aktuelle Implementierung nicht effizient genug in der Vermeidung lokaler Minima. Sie schafft zum Beispiel den Selbstalignment-Test nicht, sondern erreicht für die meisten Eingabegraphen nur etwa 85% der Optimallösung.

Netzwerkalignment durch evolutionären Algorithmus (EA). In Kapitel 9 wird GEDEVO beschrieben, ein neuer Algorithmus zur Lösung des Alignmentproblems für Paare von Netzwerken. GEDEVO nutzt einen evolutionären Algorithmus um die GED zu optimieren. Die wichtigsten Vorteile dieses Ansatzes sind seine Einfachheit und Flexibilität. Wir führen elementare evolutionäre Operationen auf einer Abbildung aus. Dank des GED-Modells können diese von allen möglichen relevanten Daten zusätzlich unterstützt werden. Zu den Nachteilen von GEDEVO gehören lange Laufzeiten und hoher Speicherbedarf. Auf echten Daten wird demonstriert, dass dieser Ansatz vergleichbar ist mit anderen aktuellen Methoden zum Netzwerkalignment und diese sogar oft übertrifft, obwohl nur topologische Informationen verwendet werden. Außerdem wird gezeigt, dass unser Ansatz ein perfektes Selbstalignment findet, ganz im Gegensatz zu anderen Methoden. Diesen Test zu bestehen sollte eine Mindestvoraussetzung für jeden Netzwerkalignmentalgorithmus sein.

Mehrfach-Metzwerkalignment durch evolutionären Algorithmus. In Kapitel 10 wird GEDEVO-M, eine Heuristik zum Alignment mehrerer Netzwerke beschrieben. Dafür wird die gesamte GED über mehrere Netzwerke als Optimierungskriterium definiert und der evolutionäre Algorithmus aus Kapitel 9 angewendet. Entsprechend werden die evolutionären Operatoren von GEDEVO erweitert und die Methode sowohl mit eukaryotischen als auch mit bakteriellen PPI-Netzwerken getestet, ebenfalls unter alleiniger Verwendung von topologischen Informationen. Wir zeigen außerdem, dass GEDEVO-M den Selbstalignment-Test besteht, indem es mehrere identische Kopien eines Netzwerks perfekt aligniert. Die Laufzeit hängt jedoch von der Anzahl der Eingabenetzwerke ab: Mit wachsender Anzahl von Netzwerken wächst der Suchraum exponentiell und somit auch die von GEDEVO-M benötigte Zeit, mehrere identische Kopien eines Netzwerks zu alignieren. Um die Methode besser auf größere Netzwerke anwenden zu können, sollte die zur Laufzeit benötigte Zeit verbessert werden, zum Beispiel durch die Begrenzung des Suchraums mit Hilfe zusätzlicher relevanter biologischer Daten.

Abstract

In the thesis we aim to study theoretical and practical questions of applying the graph edit distance (GED) model to the protein-protein interaction network alignment problem using topological information of graphs only. In Part II we explore some theoretical aspects of the model formulated as three different problems; Part III presents three heuristics for the PPI network alignment problem based on a GED model that counts the number of deleted and inserted edges. In the following we summarize the major contribution of the work.

Covering Tree with Stars (CTS). In Chapter 3 we study the complexity border of one of the simplest formulations of the Graph Edit Distance problem – Tree Edit Distance with edge Insertions and Deletions (TED-ID). Here, we consider a special case of the problem and reformulate it as CTS. CTS asks if, given a tree and a set of stars, we can connect the input stars by adding edges between them such that the resulting tree is isomorphic to the input tree. The main result of Chapter 3 is the proof that CTS is NP-hard. From the proof it also follows that TED-ID is also NP-hard, even if the diameter of the input trees is bounded by 10. We also show that for CTS with the size of the largest star being bounded by a constant k the problem becomes polynomial-time solvable.

Neighborhood-Preserving Mapping (NPM) on trees. In Chapter 4 we introduce a new generalization of the graph isomorphism problem, called NPM. The input of NPM consists of two graphs and three integers l, d and k. The problem asks if the first graph without at most k vertices, called isolation vertices, can be mapped to the second graph. Using parameters l and d, for each mapped vertex v in the first graph we specify which neighbors of v (vertices within distance l from v) have to be mapped close enough (within distance d) to the image of v in the second graph. In NPM, the graphs with smaller size of the isolation set are thought to be more similar as less edit operations are required. We study NPM on trees and as the major result of the chapter we present a dichotomy of classical complexity of NPM on trees with respect to different values of l, d and k. Interestingly, NPM on trees is NP-hard even if

the mapping forces the first tree to "squeeze" into the second, i.e. for $d > l$ and $l \geq 3$. Despite the fact that all NP-hardness proofs in this thesis are based on reductions from the same problem (3-PARTITION), the key of every proof is the understanding of the structure of the trees to construct depending on the input parameters.

Compactness-Preserving Mapping (CPM) on trees. In Chapter 5 we consider another generalization of the graph isomorphism problem, called CPM. The input of CPM consists of two graphs and two integers l and d. The meaning of the parameter l is the same as in NPM. The parameter d corresponds to the total "error" that is allowed when close neighbors of a vertex v in the first graph are mapped to the second graph: Given the sum L_v of distances between v and its close neighbors, and the sum L'_v of distances between the image of v and the images of the close neighbors of v, CPM asks if there is a one-to-one mapping between the graphs such that for every v, $L'_v - L_v \leq d$. From one side CPM constrains NPM formulation with $k = 0$ and relaxes it from another side: In contrast to NPM it is possible that some vertices are mapped at distance greater than d, but most of the vertices have to be closer to each other. Two graphs are thought to be more similar if there is a mapping as solution for CPM with a smaller value of d. In Chapter 5 we study CPM on trees and present a dichotomy of classical complexity of CPM with respect to different values of l and d.

Network Alignment with Bee Colony Optimization. From a practical side, despite the fact that computing GED is NP-hard, efficient heuristics are required that are able to approximate an optimal solution. In Chapter 8 we describe the first heuristic presented in the thesis, called NABEECO. Approximating GED, NABEECO adopts an artificial bee colony strategy to traverse the search space and refine its solutions. We demonstrate the performance of NABEECO on a set of real protein-protein interaction data. Despite the relatively good performance of the strategy, its current implementation is not efficient enough in escaping local minima. For example it failed to pass the self-alignment test, resulting in about 85% of the optimal solution on most input graphs.

Network Alignment with Evolutionary Algorithm. In Chapter 9 we present a novel algorithm, called GEDEVO, for the pairwise network alignment problem. GEDEVO aims to optimize GED by exploiting an evolutionary strategy. The main advantages of the approach are its simplicity and flexibility: We perform elementary evolutionary operations on a mapping, which are guided by any relevant data that can be incorporated thanks to the GED model. Among its disadvantages are high running times and memory consumption. On a set of real data sets we demonstrate that our approach is comparable to and often outperforms the current tools for Network Alignment using topological information only. We show that, in contrast to other tools, our approach is also able to find a perfect alignment on a self-alignment test, which should be a requirement for any network alignment tool.

Multiple-Network Alignment with Evolutionary Algorithm. In Chapter 10 we present a heuristic, called GEDEVO-M, for the multiple network alignment problem. Here, we define the total graph edit distance on multiple networks as an optimization criterion and apply the evolutionary strategy presented in Chapter 9. Correspondingly, we extend the evolutionary operators of GEDEVO and evaluate the approach in two settings by aligning two sets of eukaryotic and bacterial PPI networks using topological information only. We also show that GEDEVO-M is able to pass a self-alignment test where it perfectly aligns several copies of a network. However, the running time here depends on the number of input networks. Since with a growing number of input networks the search space grows exponentially, the time required by GEDEVO-M to align multiple copies of the same network also raises exponentially. For the further exploitation of the method on bigger networks, the execution time should be improved by, for example, constraining search space using relevant biological data.

Preface

This thesis summarizes the work that I have done since April 2011 on approaching the network alignment problem using the graph edit distance (GED) models. The work was done within the scope of a collaboration effort between two research groups “Efficient Algorithms for Hard Problems” and “Computational Systems Biology” lead by Jiong Guo and Jan Baumbach, respectively. The thesis consists of three parts: a common introduction, and two parts that present our findings on theoretical aspects of different models of Network Alignment and practical results on applying the GED model to Network Alignment using graph topology information only.

The initial idea of applying the graph edit distance model to biological network analysis was proposed by my scientific advisors. At the beginning of my studies I came across the protein-protein interaction (PPI) network alignment problem where the graph edit distance model has an intuitive interpretation. Here, due to evolutionary conservation, PPI networks of more closely related species should have more aligned interactions (matched edges between the graphs).

On practical side, the network alignment problem required a practical software tool that is able to work with real-world data. Since the problem is known to be NP-hard, one of the priorities was to find an efficient heuristic that would have made it practically tractable. I started experimenting with heuristics that combine multiple approaches such as graph edit distance models, greedy strategies, and A^* based search. However, despite all my efforts, all the methods I tried had one inherent problem – they did not provide reasonable scalability and hence were practically unusable for larger networks. At some point I came up with the idea of randomizing the search process by applying evolutionary and other nature inspired strategies to the problem. Even though similar approaches had been tried before in the pattern recognition area, previous works lacked efficient traversal of the search space on more general graphs (including PPI networks). With my input and constant supervision together with Maximilian Malek and Jan Martens we developed two tools, called GEDEVO and NABEECO, that exploit evolutionary and bee colony optimization strategies to approximate GED in the context of the PPI network alignment. NABEECO and GEDEVO with their comparison results to current network alignment tools were presented at the *Genetic and Evolutionary Computation Conference (GECCO'13)* and *German Conference on Bioinformatics (GCB'13)*, respectively. Next, I substantially extended GEDEVO's code, including redeveloping evolutionary operators, to make it work on multiple networks. The resulting paper on multiple network alignment was presented at the *Genetic and Evolutionary Computation Conference (GECCO'14)*.

Besides the practical aspects, it was also interesting to know if an efficient exact algorithm can be derived and where the border of the NP-hardness is. It was clear that as consequence of the the subforest isomorphism problem, the general Tree Edit Distance problem with edge Insertions and edge Deletions (TED-ID) is NP-hard. Jiong suggested to consider COVERING TREE WITH STARS (CTS) that was derived from TED-ID. In close collaboration with Jiong, we first derived a polynomial-time algorithm for CTS with fixed size of the largest star. Though, it took me longer time to prove that CTS is NP-hard, even if the diameter of the input tree is bounded by 10. I presented our finding on CTS at *the International Computing and Combinatorics Conference (COCOON'13)*.

In parallel to CTS I worked on another theoretical problem. I formulated NEIGHBORHOOD-PRESERVING MAPPING (NPM) that on one side can be seen as a graph edit distance model (isolation set) and on the other side relaxes edge-to-edge correspondence, generally required in problems on graphs. Together with Jiong we considered NPM between trees. Here, I came up with all proofs and algorithms, except the NP-hardness proofs of NPM for the cases $k > 0$ and $l > d \geq 1$, whose NP-hardness were proven by Jiong. I presented our findings at the *Algorithms and Data Structures Symposium 2013 (WADS'13)*.

Having finished with NPM, I proposed another problem that was called COMPACTNESS-PRESERVING MAPPING (CPM). Similarly to NPM on trees, it is easy to show the equivalence of CPM on trees and the tree isomorphism problem for their unrelaxed cases. However, it was interesting to know where the relaxation switches the problem to NP-hard cases. Relatively fast I came up with general ideas for tree structures in the NP-hardness proofs for CPM on trees, except the case $l = 2$ and $d = 2$. I first was able to derive NP-hardness proofs for $l = 2$ and $d = 6$, and then $l = 2$ and $d = 4$. However, most of the proofs and algorithms turned out to be technically more involving than in NPM, and it took me several months to write them up. In the algorithm for the case $l = 1$ and $d = 1$ of CPM, we did not know if it is NP-hard to resolve the configuration in the second phase of the algorithm. Interestingly, I recognized that this step can be considered as a case of the Constrained Weighted P_2-Packing on Bipartite Graphs problem, that was casually mentioned at a talk of Qilong Feng at COCOON'13. The final results on CPM were presented at the *25th Annual Symposium on Combinatorial Pattern Matching (CPM'14)*.

Acknowledgments

My studies were funded and supported by the International Max Planck Research School for Computer Science (IMPRS), the Cluster of Excellence "Multimodal Computing and Interaction" (MMCI), the Saarbrücken Graduate School of Computer Science, and the Center for Bioinformatics. I would like to express my appreciation and thanks to my supervisors Jan Baumbach and Jiong Guo for giving me a chance to pursue PhD studies and for advising me and keeping an eye on my progress and development in the field of algorithm theory and bioinformatics. I would like to take the opportunity to thank Prof. Volkhard Helms for being the chair of my defence and Dr. Marc Hellmuth for taking the role of the scientific assistant. I am very thankful to Prof. Raimund Seidel, who kindly agreed to be the third reviewer of my thesis. I am also very grateful to all the people from administrative staff that maintain such a wonderful environment for scientific studies, in particular, Jennifer Gerling, Sabine Nermerich, Mona Linn, Stephanie Jörg, Polina Quaranta, Dr. Michelle Carnell, and the members of the IT help desk. I thank all the colleagues from our group, including Peng, Richy, Josch, Anne-Christin, Nic, Chris, Max, Maha, and Jan, for their helpful discussions, sharing input, and feedback. My special thanks are to Maximilian Malek and Sanjar Karaev for proofreading my thesis, and Gulnaz Mullayanova and my parents for supporting me throughout the studies.

Contents

Part I

Introduction

1 Introduction

1.1 Motivation

With the invention of the high throughput methods for detection of interactions between proteins a great amount of data has become available to the scientific community. At the web site of the National Center for Biotechnology Information we find registered sequencing projects for >1500 eukaryotes, >8500 prokaryotes, and >3000 viruses with >8 000 000 gene sequences in total [1]. However, the genes' function is often unclear and most-widely deduced from similarities to the sequences of genes with known functions. Consequently, there is still a lack of fundamental knowledge about crucial genetic programs, the interplay of genes and their products (the proteins), their biochemical regulations and their evolutionary appearance. Very little is known about such processes as cell regulation, reproduction, differentiation, and motility in response to changing internal and external conditions. Many problems in understanding these issues have to do with networks that model the interplay of all kinds of biological entities [2]. Most widely known are transcriptional gene regulatory networks and protein-protein interaction (PPI) networks. More than 150 million binary interaction evidences from protein-protein interactions that are stored on more than 28 globally distributed servers and can be accessed through PSICQUIC [3] may serve as an example for the ongoing "data explosion".

Comparing biological networks, particularly protein-protein interaction networks from different organisms, has proven very useful for inferring biological function, besides relying on DNA sequence similarity alone [4, 5]. Several problems have been formulated to analyze and compare PPI networks: network motif discovery, protein clustering, network query, and network alignment [6, 7]. In particular, Network Alignment aims to find a node-to-node correspondence between two (or more) biological networks to identify evolutionarily and functionally conserved parts between the input networks. A quality criterion of a mapping usually reflects topological aspects and biological aspects, such as the number of matched interactions induced by a mapping of the nodes from two networks, or a similarity of the biological sequences underlying the nodes.

We can distinguish the following main features of the currently available biological networks:

- The biological networks can be modeled as various types of graphs: vertex/ edge-weighted or unweighted, directed or undirected, labeled or unlabeled. For example, PPI networks that are usually modeled as unweighted graphs, can also be represented as edge-weighted graphs by adding confidence values for interactions.

- The reconstructed networks are incomplete and noisy. It is estimated in yeast, which is one of the best studied organisms, only 50% of all interactions are known; for human this number corresponds to only 10% [8]. On the other hand some interactions that were detected are invalid.

- Besides the topological information that can be derived from the networks themselves, additional biological data from various sources requires a proper integration. For example, many network alignment tools integrate all-to-all pairwise BLAST scores or BLAST E-values from protein sequence comparisons [9]. Such values are helpful in building a proper protein-to-protein correspondence, but relying on them alone is not sufficient and can even be misleading (see for example [10]).

Taking these properties of biological networks into account, the graph edit distance model appears to be a promising approach that addresses these issues, given there exists an efficient and reliable algorithm approximating it.

1.2 Graph Edit Distance

Given two graphs, the graph edit distance (GED) is defined as the sequence of edit operations that transforms one graph to another and that has minimal editing cost. Introduced in [11], GED can be seen as a natural generalization of the string edit distance and tree edit distance to general graphs. GED requires defining a set of edit operations and the corresponding cost function. The edit operation may include deletion/insertion/substitution of edges and vertices. Substituting one vertex from the first graph with a vertex from the other graph should reflect how dissimilar these two vertices are. Thus, for proper costs of edit operations, it can be shown that GED satisfies the properties to be a metric. In contrast to the exact "isomorphism" definitions when comparing graphs, GED is flexible, error-tolerant and allows modeling networks of any nature. Unfortunately, computing an optimal editing cost for even simplest definitions of GED is NP-hard and can be done with exact methods for relatively small graphs only.

Till now, GED has been mainly exploited for classification and clustering of graphs in the context of the field of pattern recognition. The editing cost of GED there serves as a metric between graphs. Many heuristics have been proposed to approximate the cost of GED. The methods vary from relaxations of integer linear programs and spectral decomposition to artificial neural networks and simulated annealing. In contrast to relatively small (with normally up to 100 nodes end edges) networks in the pattern recognition field, biological networks often have thousands of nodes and edges. Therefore, applying GED still remains challenging and development of efficient heuristics is required.

1.3 Structure of Thesis

The thesis is divided into three parts. The current "introductory" part provides a motivation for applying the graph edit distance model to network data in bioinformatics. The other two parts present the studies that are of two sides. First, since the general formulation of GED is NP-hard, we aim to identify the complexity border for some of its simplest versions. Second, since GED comes as a general approach to compare graphs, including relatively large biological networks of various types, the development of efficient heuristics is required. Correspondingly, the second "theoretical" part introduces three problems related to GED and studies them when the input graphs restricted to trees, and in the third "practical" part we present heuristics for protein-protein interaction network alignment.

Part II starts with an overview of the results related to GED to highlight the difficulty of the problem and introduces necessary notations (Chapter 2). Chapter 3 studies one of the simplest formulations of GED with edge insertions and deletions on trees (TED-ID) and a related to it the Covering Tree with Stars problem (CTS).

In Chapters 4 and 5 we introduce two new problems, NPM and CPM, that extend the notion of edge preservation, that is common for many classical problems on graphs, such as the graph isomorphism problem. We study how the problem complexity changes depending on the degree of relaxation of the edge preservation. The last chapter of Part III discusses the results and indicates further interesting problems. The findings of Chapters 3, 4, and 5 were presented in [12] (and the journal version [13]), [14], and [15], respectively.

Part III starts with an overview of the methods and tools for Network Alignment. Then, in Chapters 8 and 9, we present two nature inspired heuristics for the pairwise global network alignment problem using topological information only. We develop two fairly general tools for Network Alignment (NABEECO and GEDEVO) that exploit a bee colony optimization strategy and evolutionary computation to align networks. Using a graph edit distance model that counts the number of deleted and inserted edges, we compare our methods to other tools for Network Alignment. In Chapter 10 we extend the definition of the pairwise global network alignment problem to multiple networks. Here, using GEDEVO's strategy, we aim to minimize the sum of pairwise graph edit distances. The last chapter of Part III indicates the possible extensions of the proposed approaches and gives some further directions for studies in the filed of network alignment. The results described in Chapters 8, 9, and 10 were presented in [16], [17], and [18], respectively.

1.4 Publications

Ibragimov, R., Malek, M., Baumbach, J., Guo, J.: Multiple graph edit distance – simultaneous topological alignment of multiple protein-protein interaction networks with an evolutionary algorithm. In: Genetic and Evolutionary Computation Conference, GECCO 2014, ACM (2014) 277–283

Baumbach, J., Guo, J., Ibragimov, R.: Compactness-preserving mapping on trees. In: Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014. Volume 8486 of LNCS., Springer (2014) 162–171

Baumbach, J., Guo, J., Ibragimov, R.: Covering tree with stars. In: Computing and Combinatorics, 19th International Conference, COCOON 2013. Volume 7936 of LNCS., Springer (2013) 373–384 (journal version: Journal of Combinatorial Optimization 29(1), (2015) 141–152)

Ibragimov, R., Malek, M., Guo, J., Baumbach, J.: GEDEVO: an evolutionary graph edit distance algorithm for biological network alignment. In: German Conference on Bioinformatics 2013, GCB 2013. Volume 34 of OASICS., Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013) 68–79

Ibragimov, R., Martens, J., Guo, J., Baumbach, J.: NABEECO: biological network alignment with bee colony optimization algorithm. In: Genetic and Evolutionary Computation Conference, GECCO 2013, ACM (2013) 43–44

Baumbach, J., Guo, J., Ibragimov, R.: Neighborhood-preserving mapping between trees. In: Algorithms and Data Structures - 13th International Symposium, WADS 2013. Volume 8037 of LNCS., Springer (2013) 427–438

Part II

Topological Models of Network Alignment

2 Background and Related Work

2.1 Tree Modification Problems

Graphs are one of the most common representations of objects and relations between them in many fields: computational biology, computational chemistry, computer vision, pattern recognition, etc. The natural need to analyze and compare such objects has stimulated the development of various indicators that reflect differences of vertices and edges and also differences in their topologies. In many applications, two graphs G_1 and G_2 are thought to be equal only if their topologies are identical. In this context, the cost in the Graph Edit Distance problem (GED) is one of the most common measures for the graphs, which has been extensively used in graph matching in particular [19]. The general definition of GED is as follows. Given two graphs, GED asks to compute a *sequence of the edit operations* that transforms one graph into another and that has the minimal *editing cost* over all possible such sequences. Depending on the application, the *set of edit operations* may include relabeling, deletion, insertion, contraction of edges and relabeling, deletion, insertion, or splitting of vertices. The cost function that maps edit operations and their arguments to nonnegative numbers is also application-specific and may, for example, depend on whether the graphs are labeled, weighted, directed, etc. The cost of an edit operation corresponds to the amount of distortion induced by the operation. Irrespective of the exact problem-specific definition of GED, the main concept behind it is that the more similar two graphs are, the closer they will be in the problem space, and the lower is the cost of the optimal edit sequence.

In one of the simplest formulations of GED, the set of edit operations contains only deletion and insertion of edges with the input graphs having the same number of vertices. Here, the evidence for the cost of the edit distance can be computed from the corresponding *one-to-one mapping* between the vertices of the input graphs. However, even with this simplest set of edit operations, application of GED is limited by its complexity since computing the edit distance between two graphs is not easier than the subgraph isomorphism problem, which is known to be NP-complete [20].

In this part of the thesis we explore some theoretical borders of complexity for three different problems related to Graph Edit Distance. We will require both input graphs to be trees since in this case many graph problems become easier. First, we consider a variation of GED with edge insertions and edge deletions as edit operations on trees (Chapter 3). Further, we relax the notion of "edge preservation" that is commonly used in graph problems: An edge is preserved if the images of two adjacent vertices in the first graph are also adjacent in the second graph. We define two new generalizations of the graph isomorphism problem where we require that for any pair of neighboring vertices their images also remain to some extent close to each other in the second graph (Chapters 4-5).

In Chapter 3 we aim to explore the complexity border of Graph Edit Distance and study the Tree Edit Distance problem with edge Insertions and edge Deletions (TED-ID). We formulate a special case of this problem as Covering Tree with Stars (CTS) that is defined as follows. Given a tree T and a set $\mathcal{S}$ of stars, CTS asks if it is possible to connect the stars in $\mathcal{S}$ by adding edges between them such that the resulting tree is isomorphic to T. We prove that in the general setting, CST is NP-complete, which implies that TED-ID is also NP-hard, even when both input trees have diameters bounded by 10. We also show that, when the number of distinct stars is bounded by a constant k, CTS can be solved in polynomial time, by presenting a dynamic programming algorithm running in $O(|V(T)|^2 \cdot k \cdot |V(\mathcal{S})|^{2k})$ time.

The study presented in Chapter 4 is motivated by the following idea. In Network Alignment, if subnetworks in two networks are conserved, then the neighborhoods of the corresponding nodes are also conserved. We formalize this idea by introducing a variation of the graph isomorphism problem, where, given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and three integers l, d, and k, we seek for an isolation set $D \subseteq V_1$ and a one-to-one mapping $f : V_1 \to V_2$, such that $|D| \leq k$ and for every vertex $v \in V_1 \setminus D$ and every vertex $u \in N^l_{G_1}(v) \setminus D$ we have $f(u) \in N^d_{G_2}(f(v))$. Here, for a graph G and a vertex v, we use $N^i_G(v)$ to denote the set of vertices that have distance at most i to v in G. We call this problem Neighborhood-Preserving Mapping (NPM). The main result of Chapter 4 is a complete dichotomy of the classical complexity of NPM on trees with respect to different values of l, d, and k. Additionally, we present an integer linear program formulation of NPM and two dynamic programming algorithms for the cases of NPM when one of the input trees is a path.

In Chapter 5 we partially constrain the formulation of NPM and consider another variation of the graph isomorphism problem. Given two graphs $G_1 =$

(V_1, E_1) and $G_2 = (V_2, E_2)$ and two integers l and d, we seek for a one-to-one mapping $f : V_1 \to V_2$, such that for every $v \in V_1$, it holds that $L'_v - L_v \leq d$, where $L_v := \sum_{u \in N^l_{G_1}(v)} \operatorname{dist}_{G_1}(v, u)$, $L'_v := \sum_{u \in N^l_{G_1}(v)} \operatorname{dist}_{G_2}(f(v), f(u))$, and $N^i_G(v)$ denotes the set of vertices which have distance at most i to v in a graph G. We call this problem Compactness-Preserving Mapping (CPM). We study CPM with input graphs being trees and present a dichotomy of classical complexity with respect to different values of l and d. CPM on trees can be solved in polynomial time only if $l \leq 2$ and $d \leq 1$. Additionally, we present an integer linear program formulation for an optimization version of CPM that, for the given parameters l and d, minimizes the size k of the isolation set; the isolation set here is defined in the same way as in NPM.

Next we provide a brief overview of the related work and introduce the necessary notation and definition that will be used throughout Part II.

2.2 Related Work

Graph Edit Distance is closely related to several graph problems and their variations, namely Subgraph Isomorphism, Maximum Common Subgraph, graph modification problems, and Graph Packing.

Graph Edit Distance. First, to give an intuition about the difficulty of GED consider one of its simplest, yet realistic cases: The input graphs are unlabeled and undirected, and the set of edit operations includes deletion and insertion of edges and vertices with unit cost per operation. Here, if the cost of GED is 0 (i.e. no vertex or edge insertions or deletions are required), then the input graphs are isomorphic. However, there is no known polynomial time algorithm for Graph Isomorphism (it is not known to be in P or NP-complete yet). The best known algorithm for Graph Isomorphism requires $O(2^{\sqrt{n \cdot \log n}})$ time [21].

If G_1 is a subgraph of G_2, then the corresponding cost of GED has the lowest value over all possible input graphs with the same number of vertices and edges; the cost here equals to $(|V_2|-|V_1|)+(|E_2|-|E_1|)$, which corresponds to inserting $|V_2| - |V_1|$ vertices and $|E_2| - |E_1|$ edges. However, Subgraph Isomorphism is NP-hard [20].

Exact algorithms for GED, such as based on A* [22] or its integer linear program formulations [23, 24] currently cannot be applied to graphs with larger number of veritces (> 100). However, even guaranteed approximate solutions can be helpful in many problems, e.g. clustering of graphs. A lower bound for the cost of GED can be obtained in polynomial time by relaxation of its integer linear program formulation. In [25], another method to obtain a lower bound for the cost was proposed: The approximation is computed by resolving a bipartite graph matching (in $O(n^3)$ time) that is built using pairwise distances between vertices of the input graphs. These distances are computed from the differences between corresponding vertices and the lowest possible differences between the incident edges (stars). As a starting point for studying approximability of GED may serve a so-called Graph Transformation (GT) problem, that given two graphs and an integer k, asks if the first graph can be made isomorphic to the second graph by relocating (inserting and deleting) of k edges. In [26] it was proved that the optimization version of GT with respect to k is APX-hard, meaning that for GT there exists a polynomial-time approximation algorithm with a performance guarantee bounded by a constant factor of the optimum, and no polynomial-time approximation scheme exists (unless P=NP). Due to the hardness of GED, many heuristics based on a wide range of techniques were proposed for practical applications (see for example [19] and [27]). We review some of the methods with applications to Network Alignment in Chapter 7.

Subgraph Isomorphism. Subgraph Isomorphism, which can be considered as a special case of GED, remains NP-hard even if the first graph is a tree and the other graph has treewidth of two [28]. Even parameterization of the problem by different values (such as maximum degree, number of vertices, number of connected components, treewidth, pathwidth, genus, etc.) leaves little hope for real-world data [29] due to high running times. A number of optimization versions of Graph Isomorphism were considered in [30] (see also [31]). In particular, it was shown that for the problem that maximizes the number of matched vertex pairs and for any constant $\alpha < 1$, there is an α-approximation algorithm running in time $n^{O(\log n/(1-\alpha)^4)}$. Another closely related to GED problem is the Maximum Common Subgraph problem (MCS) which, given two graphs, asks to find subsets $E'_1 \subseteq E_1$ and $E'_2 \subseteq E_2$, such that two graphs $G'_1 = (V_1, E'_1)$ and $G'_2 = (V_2, E'_2)$ are isomorphic and $|E'_1|$ is maximal. Indeed, the equivalence between MCS and GED with a specific cost function was shown in [32].

Clearly, on top of these results it can be seen that GED is even more complicated than Subgraph Isomorphism.

Subtree Isomorphism. In contrast to general graphs, isomorphism problems on trees are known to be polynomial-time solvable. Given (unrooted) trees T_1 and T_2, Subtree Isomorphism asks to find a subtree of T_2 which is isomorphic to T_1 or decide that there is no such subtree. One of the first algorithms solving the problem requires $O(n^{5/2})$ time [33] (see also [34]), where n is the size of input trees. The running time was improved to $O(n^{5/2}/\log n)$ in [35]. In the case of Tree Isomorphism the problem can be solved in linear time [36]. In contrast to Subtree Isomorphism, the related Subforest Isomorphism problem that, given a forest F and a tree T, asks whether F is isomorphic to T, was shown to be NP-hard [20]. Moreover, in Chapter 3 we show that the problem remains NP-hard even if the forest is a collection of stars and the height of the tree is bounded by 10.

Tree Edit Distance. Similar to Graph Edit Distance, the Tree Edit Distance problem between trees T_1 and T_2 is defined a sequence of edit operations that transforms T_1 into T_2 and has the lowest editing cost. A number of studies were focused on rooted labeled ordered and unordered trees with a set of edit operations that includes relabeling, deleting and inserting of edges and vertices. A rooted tree is ordered if there is a linear order among siblings of every vertex. Here, the operations of deletion and insertion do not decompose the tree into a forest, but preserve the hierarchy. That is, if a vertex v is deleted, then the children of v become the children of the parent of v instead of v. Oppositely, a vertex v is inserted as a child of a vertex u with (consecutive) subset of children of u; these children become the children of v. The ordered version was first shown to be solvable in $O(n^5)$ time [37], where n is the number of vertices in the input trees. The running time was consecutively improved to $O(n^4)$ in [38], then to $O(n^3 \log n)$ in [39], and finally to $O(n^3)$ in [40]. The latter running time was also shown to be worst-case optimal for the decomposition strategy algorithms. A more refined algorithm was presented and evaluated in [41]. For unordered trees the problem becomes NP-hard [42] and MAX SNP-hard [43], which means that no polynomial-time approximation scheme exists for the problem (unless P=NP). Further results on the tree edit distance problem, including parameterization and approximation algorithms, can be found for example in [44, 45, 46, 47, 48, 49, 50].

Graph Modification Problems. Another set of problems related to GED includes graph modification problems. Here "modifications" may include such operations as edge/vertex deletion or insertion, and the aim is to convert the input graph into a graph that satisfies a certain property with the minimum number of modifications. The property of the resulting graph can be transitivity, completeness, acyclicity, planarity, etc. Unfortunately, for many such properties the corresponding problems are NP-hard: A general result for edge/vertex deletion problems was presented in [51, 52]. A summary of results for edge modification problems can be found in [53] and [54].

Graph Packing and Graph Covering. Another class of problems closely related to GED consists of so-called packing problems. Given a graph G and a graph H, the H-packing problem asks to find a subset $\mathcal{G}'$ of edge-disjoint subgraphs of G, such that each element of $\mathcal{G}'$ is isomorphic to H and $\mathcal{G}'$ has maximum cardinality. Another related problem is called $\mathcal{H}$-covering, which, given a graph G and a set of graphs $\mathcal{H}$, covers the vertices of G by a minimum number of copies of graphs from $\mathcal{H}$, such that every vertex of G is covered exactly once. In [55] it was proven that the H-packing packing (as well as the $\mathcal{H}$-covering problem) is NP-complete even if H is a graph with three connected vertices. Note, the classical matching problem, i.e., when H is a path of length one, the problem is solvable in polynomial time [56, 57]. Different versions of the covering problem where all graphs are rooted trees were studied in [58].

2.3 Preliminaries

In the following we provide notations and definitions used throughout Part II.

Graphs. We consider only simple undirected graphs without self-loop. Given a graph G, we use $V(G)$ and $E(G)$ to denote the vertex and edge sets of G, respectively. The number of vertices in a graph G is denoted by $|G|$ or $|V(G)|$.

We call a graph $P = (V(P), E(P))$ a *path* if there is a permutation of the vertices $i_1, \dots, i_{|V(P)|}$, such that $(v_{i_{k-1}}, v_{i_k}) \in E(P)$, for $k = 2, \dots, |V(P)|$, and $|E(P)| = |V(P)| - 1$ (the *length* of the path). The vertices of degree 1 in the path are called the *end-vertices*. A path in graph G is a subgraph of G which is a path. By $\text{path}_G(v, u)$ we denote a shortest path connecting vertices v and u in a graph G.

The (*direct*) *neighborhood* of a vertex v in a graph G, denoted by $N_G(v)$, is the set of vertices which are adjacent to v. The *degree* of v is $|N_G(v)|$. We use $N_G[v]$ to denote $N_G(v) \cup \{v\}$. The *distance* $\text{dist}_G(u, v)$ between two vertices $u, v \in V(G)$ is the length of the shortest path between u and v in G. The *diameter* of a graph G is defined as the maximal distance between two vertices in G. For any integer $i > 1$, we call the set of vertices that have distance at most i to v the *i-neighborhood* of v, denoted by $N^i_G(v)$. The *exact i-neighborhood* $\hat{N}^i_G(v)$ of vertex $v \in G$ is a set of vertices that have the distance of exactly i to v in G.

A *forest* is a graph without cycles, while a *tree* is a connected forest. The degree-one vertices of a tree are called the leaves, and the others are the internal vertices. A *subtree* is a connected subgraph of a tree. A *rooted tree* T is a tree with one vertex designated to be the root of T. Given a rooted tree T, $T(v)$ denotes the subtree consisting of a vertex $v \in V(T)$ and all its descendants in T. For vertices u and v of an unrooted tree T, the subtree $T(u)$ with T rooted at v is denoted by $T(u, v)$. A *star* is a tree with at most one internal vertex. This internal vertex is then called the *center* of this star. The size of a star S is the number of edges in S. A singleton is an isolated vertex.

The graph P_2 is a path of length two. A *P_2-packing* of size q in a graph G is a collection of q vertex-disjoint P_2's in G.

Given two graphs G and G' and an injective mapping $f : V(G') \to V(G)$, called a *subisomorphism*, such that for all $(v_1, v_2) \in E(G')$, $(f(v_1), f(v_2)) \in E(G)$, G' is a *subgraph* of graph G (written as $G' \subseteq G$); then vertex $v \in V(G)$ is *covered* if $f^{-1}(v) \neq \emptyset$. The graph G is *covered* with the graph G' if the mapping f is bijective. An *induced subgraph* of a graph G is a graph $G' = (V', E')$, with $V(G') \subseteq V(G)$ and $E(G') = \{(v,u) : v, u \in V(G')\} \cap E(G)$. Graphs G and G' are *isomorphic* if there exists a one-to-one mapping $f : V(G) \to V(G')$, such that $(u,v) \in E(G)$ iff $(f(u), f(v)) \in E(G')$.

Complexity. We use the standard "big-Oh" notation to denote the asymptotic upper bound for the running time of algorithms.

If $g(n)$ is the worst running time of an algorithm with an input of size n and $f(n)$ is a function such that there exist constants $c > 0$ and $n_0 > 0$ satisfying $g(n) \leq c \cdot f(n)$ for all $n \geq n_0$, then we write $g(n) = O(f(n))$.

The NP-hardness of problems considered in the thesis is proven by reductions from the following problem:

3-Partition
Input: A multiset A of $3n$ integers $a_1, \ldots, a_{3n}$ and an integer B.
Question: Can A be partitioned into n subsets such that each subset contains exactly three integers and the sum of the integers in each subset is B?

In [20] it was shown that 3-Partition remains NP-hard even when every integer in A is bounded by a polynomial in n.

3 Covering Tree with Stars

Restricting general graph in Graph Edit Distance to trees branched in investigating variations of the tree edit distance problem. In this chapter we aim at exploring the complexity of computing the edit distance between trees with the edit operations affecting only the edges, namely, deleting and inserting edges:

> Tree Edit Distance with Edge Insertion and Deletion (TED-ID)
> **Input:** Trees T_1 and T_2, a non-negative integer d.
> **Question:** Can we modify T_1 by adding or deleting at most d edges in it such that the resulting tree is isomorphic to T_2?

TED-ID is partially motivated by alignment of backbone trees extracted from protein-protein interaction networks of different species. Here, large continuous regions of aligned subtrees can serve as strong indication for biologically meaningful node-to-node correspondence, easing at the same time matching of the other, not matched, nodes of the networks.

Given a solution for TED-ID, we can perform the edit operations in the order of first deleting edges from the first tree, resulting in a forest, and then inserting other edges to connect the forest. Thus, the problem of transforming a given forest to a given tree by deleting zero edges and adding a number of edges can be considered as a separate phase of the tree edit distance problem. We observe that, the NP-completeness of the subforest isomorphism problem [20] implies that this edge addition problem is NP-complete. By a simple reduction from 3-Partition [20], we can even prove the NP-hardness for the forest being a collection of paths.

Motivated by these NP-hardness results, we consider another restriction on the components of the forest, that is, all components being stars, and intend to find a boundary of the complexity for the tree edit distance problem:

> Covering Tree with Stars (CTS)
> **Input:** A collection of stars $\mathcal{S}$ and a tree T.
> **Question:** Can we connect the stars in $\mathcal{S}$ by adding edges between them such that the resulting tree is isomorphic to T?

One can consider CTS as using the stars to cover the vertices of the tree. We use $|\mathcal{S}|$ to denote the number of stars in $\mathcal{S}$ and k the number of distinct stars in $\mathcal{S}$. Moreover, let $V(\mathcal{S})$ and $V(T)$ be the set of the vertices in the stars of $\mathcal{S}$ and T, respectively. Clearly, we can assume $|V(\mathcal{S})| = |V(T)|$. Moreover, the number of edges added to the stars is exactly $|\mathcal{S}|-1$. CTS can also be formulated as a matching problem between $\mathcal{S}$ and T. Here, one is seeking for a one-to-one mapping f from $V(\mathcal{S})$ to $V(T)$ such that f "preserves" the edges in $\mathcal{S}$, that is, if $u, v \in V(\mathcal{S})$ are adjacent, then $f(u)$ and $f(v)$ must be adjacent. We obtain a classification of the complexity of CTS with respect to the number k of distinct stars in $\mathcal{S}$: CTS can be solved in polynomial time, if k is bounded by a constant; otherwise, this problem is NP-complete. The latter is shown by a reduction from 3-Partition. As a corollary of this NP-complete result, we show that TED-ID remains NP-hard, even when both trees have bounded diameters. CTS resembles the subgraph packing problem, which, given graphs G and H, asks to find the maximum number of vertex-disjoint subgraphs in G each of which is isomorphic to H. However, the problem is NP-hard for any H with more than two vertices [55]. My contribution here is the NP-hardness proof for CTS and the polynomial time algorithm for CTS when the size of the largest star is bounded by a constant.

3.1 NP-Completeness Results

We first show that the edge addition problem to transform a forest to a tree is NP-hard even for a forest being a collection of paths. Then we show the NP-hardness of the general setting of Covering Tree with Stars (CTS), that is, there are unbounded many distinct stars in $\mathcal{S}$. As a corollary of this hardness, we prove that TED-ID is NP-hard, even when both trees have bounded diameters.

Theorem 1. *Given a collection of paths F and a tree T, transforming F to T by edge additions is NP-hard.*

Proof. To prove the NP-hardness, we reduce from the 3-Partition problem.

Given an instance (A, B), the corresponding collection F consists of $3n$ paths of length $a_i - 1$ for $i = 1, \ldots, 3n$. Every path corresponds to one distinct element of A. There is also one additional path p of length $2B$. The corresponding tree T is composed of $n+2$ paths of length $B-1$ connected with $n+2$ edges to one additional vertex r; for each of these paths, there is an edge between r and one of its end-vertices.

To connect all $3n+1$ paths from F by $3n$ edge additions, we are forced to map the middle vertex of path p to vertex r in the tree. The other vertices of p have to be mapped to two paths of length $B-1$ in T. It is not hard to see that the remaining $3n$ paths from F can be mapped to unmapped vertices in T, if and only if there is a 3-partition of set A. □

Theorem 2. *CTS is NP-complete.*

Proof. Clearly, CTS is in NP, since we can guess $|\mathcal{S}|-1$ edges between the stars in $\mathcal{S}$ in polynomial time and the isomorphism testing between two trees can be done in polynomial time [59]. We reduce again from 3-Partition.

Given an instance (A, B), we can safely assume that $0 \leq a_i < B$ for each $i = 1, \ldots, 3n$. We construct an instance $(\mathcal{S}, T)$ of CTS in the following way. First, $\mathcal{S}$ consists of five subcollections of stars. The first one has only one "large" star, denoted as S_L, which has $n^4 \cdot B + n$ leaves. The second subcollection $\mathcal{S}_2$ contains $3n$ stars, one-to-one corresponding to the integers in A. For each integer a_i with $i = 1, \ldots, 3n$, we add a star with $n^3 \cdot B + a_i$ many leaves to $\mathcal{S}_2$. The third subcollection $\mathcal{S}_3$ contains $2B \cdot n$ stars. More precisely, for each $i = 1, \ldots, n$, we create a collection $\mathcal{S}_3^i$, which contains $2B$ stars, each having $M_i := n^2 \cdot B + i$ many leaves. We set $\mathcal{S}_3 = \bigcup_{i=1}^n \mathcal{S}_3^i$. Moreover, we create for each $i = 1, \ldots, n$ a collection $\mathcal{S}_4^i$ with $B \cdot M_i$ stars, each having $N_i := n \cdot B + i$ leaves, and set $\mathcal{S}_4 := \bigcup_{i=1}^n \mathcal{S}_4^i$. Finally, the last subcollection $\mathcal{S}_5$ consists of $\sum_{i=1}^n (n \cdot 2B \cdot M_i \cdot N_i)$ many "singletons".

Next, we construct the tree T. To ease the description, we describe T as a tree rooted at a vertex r, which has $n^4 \cdot B$ many leaves as children (see also Fig. 3.1 for the depiction of tree T). In addition, r has n children $r_1, \ldots, r_n$, which are the roots of n subtrees, denoted as $T_1, \ldots, T_n$. Let R_1 be the set of r's children. For each $i = 1, \ldots, n$, the vertex r_i has three children r_i^l with $l = 1, 2, 3$. Let R_2 be set containing all $3n$ children of the non-leaf vertices in R_1. Each vertex $r_i^l \in R_2$ has $n^3 \cdot B$ leaves as children. In addition, r_i^l has other B children $r_i^l[j]$ with $j = 1, \ldots, B$, at which the subtrees $T_i^l[j]$ are rooted. Let R_3^i be the set of the non-leaf children $r_i^l[j]$ of r_i^l for all $l = 1, 2, 3$. Clearly, $|R_3^i| = 3B$. We set $R_3 := \bigcup_{i=1}^n R_3^i$. Finally, in each $T_i^l[j]$, $r_i^l[j]$ has $M_i = n^2 \cdot B + i$ many children, each of which has again $N_i = n \cdot B + i$ many leaves as children. The set R_4^i contains all children of the vertices in R_3^i, that is, $|R_4^i| = 3B \cdot M_i$. We set $R_4 := \bigcup_{i=1}^n R_4^i$. This completes the construction of T.

Since every integer and thus B are bounded by a polynomial of n, the construction of $(\mathcal{S}, T)$ clearly needs polynomial time. Next, we prove that (A, B) is a yes-instance iff $(\mathcal{S}, T)$ is a yes-instance.

"$\Longrightarrow$": Let $A_1, \ldots, A_n$ be a partition of A with $|A_i| = 3$ and $\sum_{a \in A_i} a = B$ for each $i = 1, \ldots, n$. A mapping f between $V(\mathcal{S})$ and $V(T)$ can be derived as follows. First, map the center of the large star S_L to the root r of T and S_L's leaves to the children of r. Then, for each $i = 1, \ldots, n$, let $A_i = \{a_i^1, a_i^2, a_i^3\}$. For each $l = 1, 2, 3$, we map the center of the star S in $\mathcal{S}_2$, which corresponds to a_i^l, to the child r_i^l of r_i. Note that r_i is a child of r and the root of the

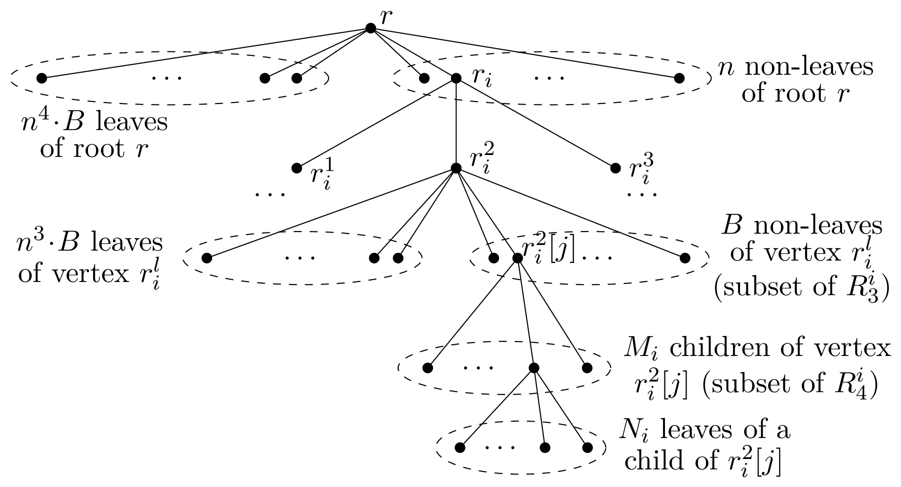

r
n non-leaves of root r
r_i
$n^4 \cdot B$ leaves of root r
r_i^1
r_i^2
r_i^3
$n^3 \cdot B$ leaves of vertex r_i^l
$r_i^2[j]$
B non-leaves of vertex r_i^l (subset of R_3^i)
M_i children of vertex $r_i^2[j]$ (subset of R_4^i)
N_i leaves of a child of $r_i^2[j]$

Fig. 3.1: The tree T of CTS corresponding to an instance of 3-PARTITION in the proof of Thm. 2

subtree T_i. Moreover, $n^3 \cdot B$ many leaves of S are mapped to the $n^3 \cdot B$ leaf children of r_i^l. The remaining a_i^l leaves of S are then mapped to the non-leaf children of r_i^l. Note that r_i^l has now $B - a_i^l$ unmapped children and r_i^1, r_i^2, r_i^3 have together $2B$ unmapped children. Let E be the set of unmapped children of r_i^1, r_i^2, r_i^3 and $D := R_3^i \setminus E$. Clearly, $E \subseteq R_3^i$. Now, we map the centers of the stars in $\mathcal{S}_3^i$ to the vertices in E. Since each of the vertices in E has exactly $M_i = n^2 \cdot B + i$ many children, the leaves of the stars in $\mathcal{S}_3^i$ can be mapped to the children of the vertices in E. By this way, all stars in $\mathcal{S}_3$ can be mapped to the tree. Note that each of the children of the vertices in E has $N_i = n \cdot B + i$ leaves as children, which are still unmapped. Now consider the stars in $\mathcal{S}_4$. Recall that $|\mathcal{S}_4^i| = B \cdot M_i$ and each star in $\mathcal{S}_4^i$ has N_i leaves. Note that the vertices in D are mapped to the leaves of the stars in $\mathcal{S}_2^i$ and each vertex in D has M_i many children, each of which has again $n \cdot B + i$ leaves as children. By $|D| = B$, we can conclude that the stars in $\mathcal{S}_4^i$ can be mapped to the subtrees rooted at the children of the vertices in D. Finally, recall that, in the above analysis, each of the children of the vertices in E has $N_i := n \cdot B + i$ unmapped leaves as children. Thus, in each subtree T_i there are exactly $2B \cdot M_i \cdot N_i$ many unmapped leaves. Summing up over all T_i's, the singletons in $\mathcal{S}_5$ can then be mapped. Since we always map the leaves of a star together with its center, the mapping clearly satisfies the edge-preserving condition.

"$\Longleftarrow$": Let f be a mapping from $V(\mathcal{S})$ to $V(T)$. In order to prove this direction, we need the following claims. The first three claims follow directly from the degrees of the root and the vertices in R_2 and R_3.

Claim 2.1: The center of the large star S_L has to be mapped to the root r of T, and the leaves of S_L one-to-one to the vertices in R_1.

Claim 2.2: The centers of the stars in $\mathcal{S}_2$ have to be mapped one-to-one to the vertices in R_2.

Claim 2.3: The centers of the stars in $\mathcal{S}_3$ have to be mapped to the vertices in R_3.

By Claim 2.2, the stars in $\mathcal{S}_2$ and thus the integers in A are partitioned by the mapping f into n subsets, denoted by $A_1, \ldots, A_n$, such that $|A_i| = 3$ for all $i = 1, \ldots, n$. It remains to prove that $\sum_{a \in A_i} a = B$ for all $i = 1, \ldots, n$. To this end, we need the following claim, which is true, since a star in $\mathcal{S}_2$ has more leaves than the number of leaf children of a vertex in R_2 and mapping a leaf in T to a singleton is never better than mapping it to a leaf of a star in $\mathcal{S}$.

Claim 2.4: If $(\mathcal{S}, T)$ is a yes-instance, then there is a mapping such that the leaf children of the vertices in R_2 are all mapped to the leaves of the stars in $\mathcal{S}_2$.

According to Claims 2.3 and 2.4, $2nB$ vertices in R_3 have to be mapped to the centers of the stars in $\mathcal{S}_3$ and the remaining vertices have to be mapped to the leaves of the stars in $\mathcal{S}_2$. Since we can w.l.o.g. assume $\sum_{a \in A} a = nB$, the following claim holds.

Claim 2.5: The centers of the stars in $\mathcal{S}_4$ have to be mapped to the vertices in R_4.

In order to prove $\sum_{a \in A_i} a = B$ for all $i = 1, \ldots, n$, we apply a backward induction from $i = n$ to $i = 1$. For $i = n$, let $A_n = \{a_n^1, a_n^2, a_n^3\}$ and let S_n^1, S_n^2, S_n^3 denote the three stars in $\mathcal{S}_2$, which correspond to the integers in A_n and whose roots are mapped to r_n^1, r_n^2, r_n^3, respectively. By Claim 2.4, totally $\sum_{j=1}^{3} a_n^j$ leaves of the stars S_n^1, S_n^2, S_n^3 have to be mapped to the vertices in R_3^n. Moreover, Claim 2.3 and the fact that the stars in $\mathcal{S}_3^n$ have more leaves than all other stars in $\mathcal{S}_3^j$ with $j < n$ imply that the centers of the stars in $\mathcal{S}_3^n$ have to be mapped to the vertices in R_3^n. With $|\mathcal{S}_3^n| = 2B$ and $|R_3^n| = 3B$, we know $\sum_{j=1}^{3} a_n^j \leq B$. Let D be the set of vertices in R_3^n which are mapped to the leaves of S_n^1, S_n^2, S_n^3, E_1 be the set of vertices in R_3^n that are mapped to the centers of the stars in $\mathcal{S}_3^n$, and $E_2 := R_3^n \setminus (D \cup E_1)$ (see Fig. 3.2). Clearly, $|D| = \sum_{j=1}^{3} a_n^j$ and $|E_2| = B - |D|$. If $E_2 \neq \emptyset$, then the vertices from E_2 can be mapped to the centers of other stars. However, by Claims 2.3 and 2.4, all E_2-vertices have to be mapped to the centers of some stars from $\bigcup_{j=1}^{n-1} \mathcal{S}_3^j$, since $\sum_{a \in A} a = nB$ implies that $\sum_{j=1}^{3} a_i^j > B$ for some $i < n$. Now, consider the stars in $\mathcal{S}_4^n$. Since all vertices in $S_n^1 \cup S_n^2 \cup S_n^3$ are mapped and the stars in $\mathcal{S}_4^n$ have more leaves than the degrees of the vertices in R_4^j with $j < n$, the centers of these stars can only be mapped to the vertices in R_4^n. However, by mapping the centers of the

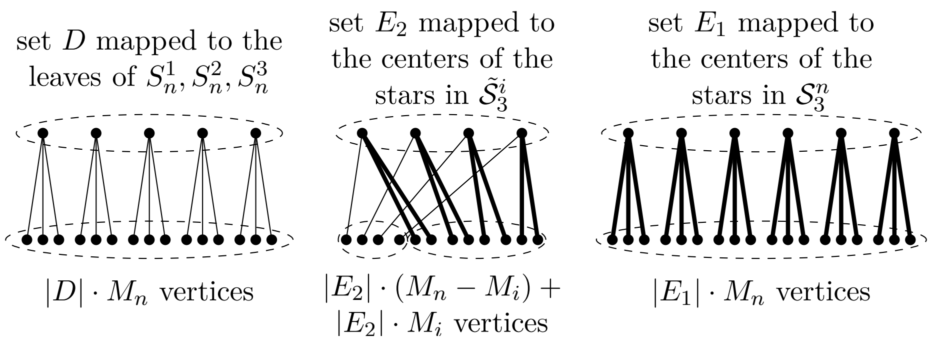

set D mapped to the leaves of S_n^1, S_n^2, S_n^3
set E_2 mapped to the centers of the stars in $\tilde{\mathcal{S}}_3^i$
set E_1 mapped to the centers of the stars in $\mathcal{S}_3^n$
$|D| \cdot M_n$ vertices
$|E_2| \cdot (M_n - M_i) +$ $|E_2| \cdot M_i$ vertices
$|E_1| \cdot M_n$ vertices

Fig. 3.2: The vertices of sets D, E_1 and E_2 in R_3^n and their children in the induction step of the proof of Thm. 2

stars in $\mathcal{S}_3^n$ to the vertices in E_1 and the centers of some stars $\tilde{\mathcal{S}}_3^i$ from $\mathcal{S}_3^i$ for some $i < n$ to the vertices in E_2, some of the vertices in R_4^n are already mapped by the leaves of these stars. For the vertices in E_1, all their children are mapped. Moreover, since $|D|+|E_2| = B$ and at least M_i children of each of the E_2-vertices are mapped to the leaves of $\tilde{\mathcal{S}}_3^i$, we have at most $|D| \cdot M_n + |E_2| \cdot (M_n - M_i) = B \cdot M_n - |E_2| \cdot M_i < B \cdot M_n$ vertices in R_4^n not mapped to the stars in $\mathcal{S}_3^n$. Recall that every vertex in R_3^n has $M_n = n^2 \cdot B + n$ many children. However, by $|\mathcal{S}_4^n| = B \cdot M_n$, the centers of some stars in $\mathcal{S}_4^n$ cannot be mapped to the vertices in R_4^n, a contradiction. Thus, we have $B = |D|$ and $\sum_{j=1}^3 a_n^j = B$. This means that $E_2 = \emptyset$ and all vertices in $R_3^n \cup R_4^n$ are mapped to stars in $\mathcal{S}_3^n \cup \mathcal{S}_4^n$. Thus, the induction step from $i+1$ to i can be conducted in a similar way as for the case $i = n$. The reason for this is that the centers of all stars in $\mathcal{S}_3^i$ can only be mapped to the vertices in R_3^i. This holds also for the stars in $\mathcal{S}_4^i$ and the vertices in R_4^i. In summary, we can derive a partition from the mapping f such that every subset A_i has exactly three integers, and $\sum_{a \in A_i} a = B$ for each $i = 1, \dots, n$. $\square$

Corollary 3. *TED-ID is NP-hard even when the diameters of both trees are bounded by 10.*

Proof. We construct an equivalent instance of TED-ID from the CTS-instance constructed in the proof of Thm. 2.

Let x be the number of the stars in the CTS-instance $(\mathcal{S}, T)$ in the proof of Thm. 2. We construct tree T_1 as follows. First, create x "big stars", each with $n^8 \cdot B^3$ leaves. Then, connect the stars in $\mathcal{S}$ one-to-one to the big stars: For each star in $\mathcal{S}$, add an edge between its center and one of the leaves of the corresponding big star. Finally, create a star with n^4 leaves, and add an edge between the center of this star and each of the centers of the big stars. Clearly, the resulting tree T_1 has a diameter equal to 8.

The second tree T_2 is firstly set equal to the tree T of the CTS-instance from the proof of Thm. 2. Then, add x "big" stars, each having $n^8 \cdot B^3$ leaves. Finally, add an edge between the root r of T and each of the centers of the big stars. The diameter of the resulting tree T_2 is clearly equal to 10.

We set $d := 2 \cdot (x-1)$. From the construction of $(\mathcal{S}, T)$, we have $d = O(n^5 \cdot B^3)$. To prove the equivalence between the constructed instance (T_1, T_2, d) and $(\mathcal{S}, T)$, observe that the big stars in T_1 can only be mapped one-to-one to the big stars in T_2. The reason for this is that the size of these stars is much greater than the allowed number d of editions. Then, we have to "separate" the stars in $\mathcal{S}$ from the big stars in T_1, and map them into the n subtrees of T_2, which correspond to the n partitions of the 3-PARTITION-instance. □

3.2 CTS with Bounded Distinct Stars

The NP-hardness of CTS motivates the study of the special case in which the number of distinct stars in $\mathcal{S}$ is bounded by a constant k. Let $S_1, \ldots, S_k$ be the distinct stars in $\mathcal{S}$. Then, $\mathcal{S}$ can be denoted by a set of pairs, that is, $\mathcal{S} = \{(S_1, n_1), \cdots, (S_k, n_k)\}$, where n_i for $i = 1, \ldots, k$ is the number of copies of S_i in $\mathcal{S}$. Moreover, we use $|S_i|$ to denote the number of edges in S_i and assume that $|S_1| < |S_2| < \cdots |S_k|$. Clearly, $|V(T)| = \sum_{i=1}^{k}(n_i \cdot (|S_i| + 1))$.

We present in the following a dynamic programming based algorithm solving CTS in polynomial time, if the number of distinct stars is bounded by a constant. The algorithm follows a bottom-up approach to process the vertices in T. Assume T is rooted at an arbitrary vertex r. For each vertex $v \in V(T)$, $T(v)$ denotes the subtree of T rooted at v. During the bottom-up process, we collect some information at every vertex v. Hereby, we firstly distinguish two cases: v is "covered" or "free". We say v is covered, if v should be mapped to the center of a star or to a leaf of a star whose center is mapped to a child of v. A vertex v is free, if v should be mapped to a leaf of a star whose center is mapped to v's parent. Note that a singleton in $\mathcal{S}$, that is, an isolated vertex, has only one center but no leaf. Concerning a star S with only one edge, we say v is free if the other vertex of S should be mapped to v's parent; otherwise, v is covered. If v is covered with v being mapped to the center of a star S, then we further distinguish some cases by the size of S. Hereby, notice that the star S does not have to be from $\mathcal{S}$ but has at most $|S_k|$ leaves. The reason for this is that the parent of v could be mapped to a leaf of S. Note that S_k is the star in $\mathcal{S}$ with the largest number of leaves.

We use C_f to denote the case of v being free and C_l to denote the case that v is mapped to a leaf of a star whose center is mapped to a child of v. Moreover, C_c^i with $i = 0, \ldots, |S_k|$ denotes the case that v is mapped to the center of a star with i leaves, which are mapped to the children of v. The three cases of

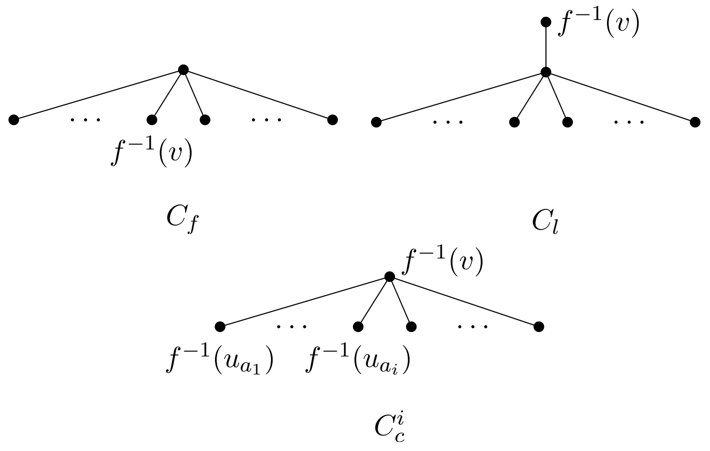


Fig. 3.3: The three cases of mapping a vertex $v \in V(T)$ to a star $S_j \in \mathcal{S}$ in CTS with bounded distinct stars

mapping vertex v and its children $u_1, \ldots, u_i$ to a star S_j are shown in Fig. 3.3. Altogether, there are $|S_k| + 3$ many cases to consider.

For each of these cases, we store all possible "realizable configurations" for v. A *configuration* is defined as a vector $K = (c_1, \ldots, c_k)$ with $c_i \leq n_i$ for all $1 \leq i \leq k$. Each vector K uniquely represents a subcollection $\mathcal{S}'$ of $\mathcal{S}$, that is, $\mathcal{S}' = \{(S_1, c_1), \ldots, (S_k, c_k)\}$. The number of all possible configurations is clearly bounded by $O(|\mathcal{S}|^k)$. We say a configuration K is "realizable" at vertex v with case C_f (or C_l), if there exists a one-to-one mapping f from $V(\mathcal{S}')$ to $V(T(v)) \setminus \{v\}$ (or $V(T(v))$) such that, if $u, v \in V(\mathcal{S}')$ are adjacent, then $f(u)$ and $f(v)$ are adjacent. In the case C_c^i, configuration K is realizable, if there exist a size-$(i+1)$ set V' consisting of v and i children of v and a mapping f from $V(\mathcal{S}')$ to $V(T(v)) \setminus V'$ with the edges in $\mathcal{S}'$ being preserved. Thus, the given instance is a yes-instance, if and only if at the root r, the configuration $(n_1, \ldots, n_k)$ is realizable with state C_l or $(n_1, \ldots, n_i - 1, \ldots, n_k)$ is realizable with state $C_c^{|S_i|}$ for some $i = 1, \ldots, k$.

The algorithm. In order to compute the realizable configurations for every vertex v and every case α, we define the following configuration sets. The set $\mathcal{K}(v, \alpha)$ should contain all realizable configurations at vertex v if the case α applies to v. Moreover, we define $\mathcal{K}(v)$ to be the set of the following configurations:

- all configurations in $\mathcal{K}(v, C_l)$,
- for each configuration $K = (c_1, \ldots, c_k)$ in $\mathcal{K}(v, C_c^i)$ with $i = |S_\alpha|$ for a star $S_\alpha \in \mathcal{S}$, the configuration $(c_1, \ldots, c_\alpha + 1, \ldots, c_k)$.

In other words, $\mathcal{K}(v)$ contains all configurations, each of which corresponds to a subcollection $\mathcal{S}'$ such that there exists a mapping f from $V(\mathcal{S}')$ to $V(T(v))$ such that the edges in the stars in $\mathcal{S}'$ are preserved. We define an addition operation on two configurations $K^1 = (c_1^1, \ldots, c_k^1)$ and $K^2 = (c_1^2, \ldots, c_k^2)$: $K_1 + K_2 = (c_1^1 + c_1^2, \ldots, c_k^1 + c_k^2)$, if $c_i^1 + c_i^2 \leq n_i$ for all $i = 1, \ldots, k$; otherwise, $K_1 + K_2$ is set to an all-0 vector. At the begin, $\mathcal{K}(v, \alpha)$ and $\mathcal{K}(v)$ for all vertices v and all cases α are set to empty.

At a leaf vertex v, the cases C_l and C_c^i with $i > 0$ cannot apply and the corresponding configuration sets remain empty. The case C_c^0 can apply, only if $\mathcal{S}$ contains the singleton. If so, then S_1 is the singleton and both $\mathcal{K}(v, C_c^0)$ and $\mathcal{K}(v)$ contain only one configuration $(c_1, \ldots, c_k)$ with $c_1 = 1$ and $c_i = 0$ for $i > 1$; otherwise, the two sets are empty. According to the definition of realizable configurations, $\mathcal{K}(v, C_f)$ contains only the all-0 vector.

Suppose we arrive at an internal vertex v with j children $u_1, u_2, \ldots, u_j$. We distinguish the cases C_f, C_l, and C_c^i.

The free case C_f. By the definition of C_f, each realizable configuration K at v has to correspond to a subcollection $\mathcal{S}'$ such that each star in $\mathcal{S}'$ has to be completely mapped to a subtree rooted at one of v's children. Moreover, every v's child u_i has to be mapped to either the center or a leaf of a star in $\mathcal{S}'$ and all other vertices of this star have to be mapped to the vertices in $T(u_i)$. Thus, every realizable configuration K can be "partitioned" into j configurations $K^1, \ldots, K^j$ such that $K^i \in \mathcal{K}(u_i)$ for every $i = 1, \ldots, j$ and $K = K^1 + \ldots + K^j$.

Based on this observation, we compute $\mathcal{K}(v, C_f)$ as follows: First, for each $i = 1, \ldots, j$, we construct $\mathcal{K}(u_i)$. To this end, we consider $\mathcal{K}(u_i, C_c^{|S_\alpha|})$ for each $\alpha = 1, \ldots, k$; for each configuration $K = (c_1, \ldots, c_k)$ of this set, we add a configuration $(c_1, \ldots, c_\alpha + 1, \ldots, c_k)$ to $\mathcal{K}(u_i)$. Finally, all configurations in $\mathcal{K}(u_i, C_l)$ are added to $\mathcal{K}(u_i)$. Then, we initiate $\mathcal{K}(v, C_f)$ as a set containing only the all-0 vector and iterate from $i = 1$ to $i = j$. For each i, we perform $K + K'$ for every pair of K and K' with K being from the old $\mathcal{K}(v, C_f)$ and $K' \in \mathcal{K}(u_i)$ and add the result to the new $\mathcal{K}(v, C_f)$.

Since the number of configurations is bounded by $O(|\mathcal{S}|^k)$, the computation of $\mathcal{K}(v, C_f)$ is doable in $O(j \cdot |\mathcal{S}|^{2k})$ time.

The covered by leaf case C_l. In this case, every configuration in $\mathcal{K}(v, C_l)$ has to correspond to a subcollection $\mathcal{S}'$, which contains one star S with one of S's leaves mapping to v and S's center mapping to a child of v. Note that S has at least two leaves. The remaining stars of $\mathcal{S}'$ have to be completely mapped to the subtrees rooted at the children of v. Therefore, one of v's children has to be of case $C_c^{|S|-1}$ and others have to be of cases C_l and C_c^i. Then, $\mathcal{K}(v, C_l)$ can be computed as follows:

$$\mathcal{K}(v, C_l) := \bigcup_{S \in \mathcal{S}, |S|>1} \left(\bigcup_{i=1,\ldots,j} \mathcal{K}^{S,i}(v, C_l) \right) \ ,$$

where $\mathcal{K}^{S,i}(v, C_l)$ contains all realizable configurations with v being mapped to a star S whose central is mapped to u_i. The set $\mathcal{K}^{S,i}(v, C_l)$ can be computed in a similar way as $\mathcal{K}(v, C_f)$: First, we compute $\mathcal{K}(u_{i'})$ for all $i' \neq i$, as in the C_f-case. We initialize $\mathcal{K}^{S,i}(v, C_l)$ as a set containing only the all-0 vector and iterate over all $i = 1, \ldots, j$ as in the C_f-case. The only exception is that the set $\mathcal{K}(u_i)$ is replaced by $\mathcal{K}(u_i, C_c^{|S|-1})$. Finally, we increase the entry of $\mathcal{K}^{S,i}(v, C_l)$ corresponding to S by one, since one copy of S is now completely mapped in $T(v)$.

Note that we only need to compute $\mathcal{K}^{S,i}(v, C_l)$ for distinct stars $S \in \mathcal{S}$. Since the computation of $\mathcal{K}^{S,i}(v, C_l)$ is basically the same as the one of $\mathcal{K}(v, C_f)$, the overall time for computing $\mathcal{K}(v, C_l)$ is bounded by $O(k \cdot j^2 \cdot |\mathcal{S}|^{2k})$.

The covered by center case C_c^i. Now, v has to be mapped to the center of a star S with $0 \leq i = |S| \leq |S_k|$ and S's leaves have to be mapped to some of $u_1, \ldots, u_j$. Note that S does not necessarily appear in $\mathcal{S}$ and $|S| \leq j$. Similarly to the C_f- and C_l-cases, all realizable configurations in $\mathcal{K}(v, C_c^{|S|})$ correspond to subcollections $\mathcal{S}'$ whose stars can be partitioned to j subsets, each being completely mapped to a subtree rooted at v's children. The only difference here is that, in order to map v to the center of S, there must be $|S|$ children of v which are "free", that is, of case C_f. Thus, it remains to compute all possible realizable configurations for every distinct star $S \in \mathcal{S}$ with $|S| \leq j$. We apply here again a dynamic programming approach. Assume that $\mathcal{K}(u_i)$ for all $i = 1, \ldots, j$ have already been computed, which can be done as described in the C_f-case.

We define a table $\mathcal{T}$ of size at most $j(j+1)/2$. The entry $\mathcal{T}[\alpha, \beta]$ with $1 \leq \alpha \leq j$ and $0 \leq \beta \leq |S| \leq j$ stores all possible realizable configurations, that we can have in the subtrees $T(u_1), \ldots, T(u_\alpha)$, if exactly β many roots of these trees are set to the free case. Clearly, we consider only $\mathcal{T}[\alpha, \beta]$ with $\beta \leq \alpha$.

The computation of the first row of $\mathcal{T}$ is trivial. By definitions, $\mathcal{T}[1, 0]$ is clearly set to $\mathcal{K}(u_1)$, while $\mathcal{T}[1, 1]$ means the same as $\mathcal{K}(u_1, C_f)$. The entries $\mathcal{T}[\alpha, \beta]$ with $\alpha > 1$ can be computed as follows:

$$\mathcal{T}[\alpha, 0] := \mathcal{T}[\alpha - 1, 0] + \mathcal{K}(u_\alpha) \ ,$$

$$\mathcal{T}[\alpha, \beta] := (\mathcal{T}[\alpha - 1, \beta] + \mathcal{K}(u_\alpha)) \cup (\mathcal{T}[\alpha - 1, \beta - 1] + \mathcal{K}(u_\alpha, C_f)) \ .$$

By the definition, the entry $\mathcal{T}[j, |S|]$ then contains all possible realizable configurations under the condition that v and $|S|$ many its children are mapped to S. Thus, we set $\mathcal{K}(v, C_c^{|S|}) := \mathcal{T}[j, |S|]$.

With the same argument as in the C_f-case, the operations between configuration sets can be done in $O(|\mathcal{S}|^{2k})$ time. Thus, the time for computing $\mathcal{T}$ is bounded by $O(j^2 \cdot |\mathcal{S}|^{2k})$.

Theorem 4. *If the number of distinct stars in $\mathcal{S}$ is bounded by a constant, then CTS can be solved in polynomial time.*

Proof. The correctness of the dynamic programming algorithm follows from the correctness of the computation for leaves and the three cases of internal vertices. From the time analysis of the cases, we can easily derive an overall $O(|V(T)|^2 \cdot k \cdot |V(\mathcal{S})|^{2k})$-time bound for the dynamic programming algorithm, which is polynomial in $|V(T)|$, if k is bounded by a constant. $\square$

4 Neighborhood-Preserving Mapping on Trees

In this chapter, we introduce a new generalized isomorphism problem. Here, we ask if there is a mapping between two graphs, such that the neighborhoods of the vertices of the first graph, except for few vertices, are preserved in the second graph. More precisely, the problem is defined as follows:

> Neighborhood-Preserving Mapping (NPM)
> **Input:** Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and three integers l, d, k.
> **Question:** Is there a set $D \subseteq V_1$ and a one-to-one mapping $f : V_1 \to V_2$ such that $|D| \leq k$ and for every vertex $v \in V_1 \setminus D$ and every vertex $u \in N^l_{G_1}(v) \setminus D$ it holds $f(u) \in N^d_{G_2}(f(v))$?

We call a solution mapping f of NPM neighborhood-preserving. Hereby, $N^i_G(v)$ denotes the set of vertices which have distance at most i to v in the graph G. The set D is called the *isolation set.*

A similar problem, called the network alignment problem, can be formulated for mappings between protein-protein interaction networks. Building a neighborhood-preserving mapping, in contrast to the classic graph isomorphism problem, provides more freedom by setting closeness constraints on the sought mapping. This freedom may help to deal with data incompleteness (missing edges or nodes) as well as noise (erroneous edges or nodes), respecting at the same time topological distance. Then, a mapping with the larger number of more important mapped nodes is thought to be more biologically meaningful.

In the chapter we focus on the classical complexity of NPM on trees, that is, both input graphs are trees. We first study NPM on trees with $k = 0$ and provide proofs for NP-hard and polynomial-time solvable cases. Table 4.1 summarizes our findings. Next we investigate the problem when $k > 0$ and prove that NPM with $k > 0$ is NP-hard for all values of l and d. Then, we present algorithms for two restricted versions of NPM on trees when one of the input trees is restricted

Tab. 4.1: Summary of the cases for NPM on trees with $k = 0$ (P stands for polynomial, NPC for NP-complete)

	$d = 1$	$d = 2$	$d \geq 3$
$l = 1$	P (Thm. 8)	NPC (Thm. 5)	NPC (Thm. 5)
$l = 2$	P (Thm. 9)	P (Thm. 16)	NPC (Thm. 5)
$l = 3$	P (Thm. 9)	P (Thm. 17)	NPC (Thm. 6)
$l \geq 4$	P (Thm. 9)	P (Thm. 17)	NPC (Thm. 7)

to be a path. We complete this chapter by providing an integer linear program formulation for the optimization version of NPM that aims to minimize the size of the isolation set D.

4.1 NPM on Trees with $k = 0$

In this section, we provide a dichotomy of the classical complexity of NPM on trees with $k = 0$; see Table 4.1 for an overview.

4.1.1 NP-Hardness Results

Case $l < d$, $d > 1$, $k = 0$

First, we show that if $k = 0$ and $1 \leq l < d$, then NPM on trees is NP-hard.

Theorem 5. *NPM on trees is NP-complete with $k = 0$, $d \geq 2$, and $l < d$.*

Proof. Clearly, NPM is in NP. To show the hardness of this case, we reduce from 3-Partition.

We first describe the reduction for NPM on trees with $k = 0$, $d = 2$, and $l = 1$ and then indicate how to extend the reduction for other cases with $1 \leq l < d$.

Given an instance (A, B) of 3-Partition, we construct an instance of NPM on trees, as depicted in Fig. 4.1: The first tree T_1 consists of $3n$ paths and a star of $2(n + 1)B + n + 2$ vertices (with the center c). Each path P_i one-to-one corresponds to an element a_i of A and contains a_i vertices. One end-vertex of each path is made adjacent to a leaf of the star, denoted by r, by adding an edge. In order to construct the second tree T_2, we first create a big star with $2(n + 1)B + 1$ vertices, n small stars, each having $B + 1$ vertices, and a special vertex r'. Then, we add edges to connect the centers of all small stars to r'. Finally, we add an edge between r' and an arbitrary leaf, denoted by c', of the big star.

To prove the equivalence between the instances, observe that the vertex c in T_1 has to be mapped to c' in T_2, since this is the only possible way to preserve

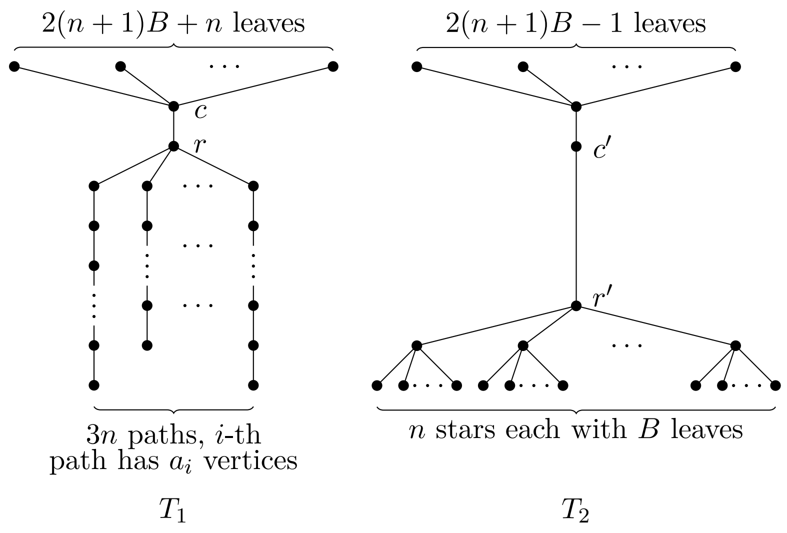

$2(n+1)B+n$ leaves
c
r
$3n$ paths, i-th path has a_i vertices
T_1
$2(n+1)B-1$ leaves
c'
r'
n stars each with B leaves
T_2

Fig. 4.1: The trees T_1 and T_2 constructed in the proof of Thm. 5

the $2(n+1)B+n+1$ neighbors of c in T_2. Thus, all vertices of the big star in T_2, except for c', are mapped to the neighbors of c. The remaining $n+1$ neighbors of c, including r, have to be mapped to r' and the centers of n small stars in T_2. Moreover, the vertex r has to be mapped to r', since r is adjacent to the end-vertices of all $3n$ paths in T_1. This implies that the vertices of the $3n$ paths have to be mapped to the leaves of the small stars in T_2. Finally, to preserve the neighborhoods of the vertices on these paths, we have to group the paths of T_1 into n groups, each of which contains 3 paths with exactly B vertices in total, corresponding to a 3-partition of the integers.

By subdividing the edges of T_1 and T_2, we can extend the reduction for other cases of d and l with $l < d$. □

Case $l \geq d$, $d \geq 3$, $k = 0$

For the case of $k = 0$ and $l \geq d \geq 3$ we prove the following two theorems:

Theorem 6. *NPM on trees is NP-complete with $k = 0$ and $l = d \geq 3$.*

Proof. We prove only the case with $l = d = 3$. The same edge subdivision idea as in the proof of Theorem 5 can be applied for $l = d > 3$. Again, we reduce from 3-Partition. Given an instance (A, B), we construct two trees as depicted in Fig. 4.2: T_1 is composed of one star (s_1) with $2nB + 2$ vertices, one star (s_2) with $n + 2$ vertices, one star (s_3) with $2n + 2$ vertices, one specific vertex r, and $3n$ encoding stars. Each encoding star corresponds to an integer in A, that is, the i-th encoding star has $a_i + 1$ vertices, for $i = 1, \ldots, 3n$. We add an edge

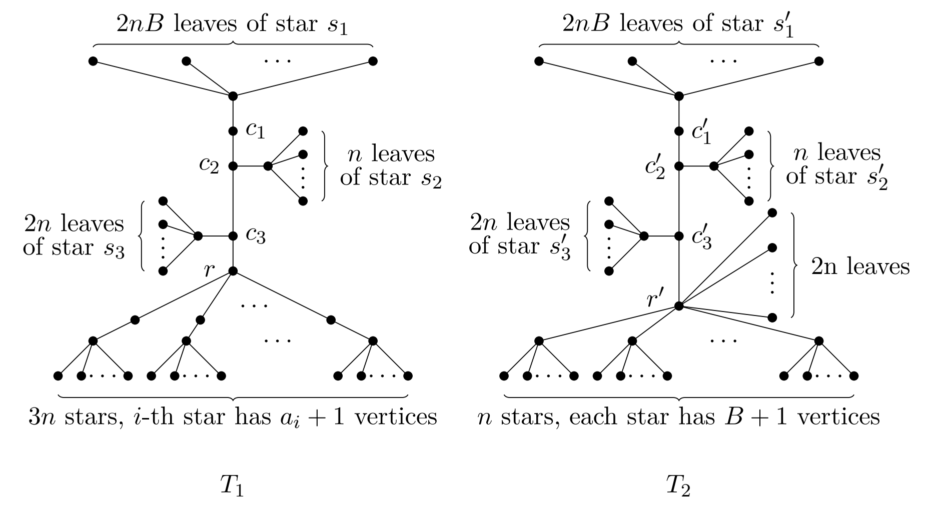

$2nB$ leaves of star s_1
c_1
c_2
n leaves of star s_2
$2n$ leaves of star s_3
c_3
r
$3n$ stars, i-th star has $a_i + 1$ vertices
T_1
$2nB$ leaves of star s'_1
c'_1
c'_2
n leaves of star s'_2
$2n$ leaves of star s'_3
c'_3
r'
2n leaves
n stars, each star has $B + 1$ vertices
T_2

Fig. 4.2: The trees T_1 and T_2 constructed in the proof of Thm. 6

to connect a leaf c_1 of star s_1 to a leaf c_2 of star s_2. Then, two edges are added to connect a leaf c_3 of star s_3 to both c_2 and r. Finally, r is connected to each of the $3n$ encoding stars by adding an edge between r and an arbitrary leaf of the star.

The corresponding tree T_2 is composed of one star (s'_1) with $2nB+2$ vertices, one star (s'_2) with $n+2$ vertices, one star (s'_3) with $2n+2$ vertices, a vertex r', and n partition stars. Each partition star has $B+1$ vertices. We add an edge between a leaf c'_1 of star s'_1 and a leaf c'_2 of star s'_2. Two edges are then added between a leaf c'_3 of star s'_3 and c'_2 and between c'_3 and r'. In addition, r' is connected to the center of every partition star. Finally, we add $2n$ leaves to T_2, which are adjacent to r'.

To prove the equivalence of the instances, observe that the vertex c_2 has to be mapped to c'_2, since $|N^3_{T_1}(c_2)| = 2nB+6n+3$ and only c'_2 has so many vertices in its 3-neighborhood. By a similar argument, c_1 has to be mapped to c'_1, and c_3 to c'_3. Therefore, the leaves of the n partition stars of T_2 can only be mapped to the $3n$ encoding stars of T_1 (except for one leaf in every encoding star). This implies the correspondence between the instances. □

Theorem 7. *NPM on trees is NP-complete with $k = 0$ and $l > d \geq 3$.*

Proof. We reduce again from 3-Partition. Let (A, B) be a 3-Partition instance. First, observe that, given a path P with $l \cdot B \cdot n^2 - 1$ vertices, there is a neighborhood-preserving mapping between P and a "squeezed" path depicted in Fig. 4.3.

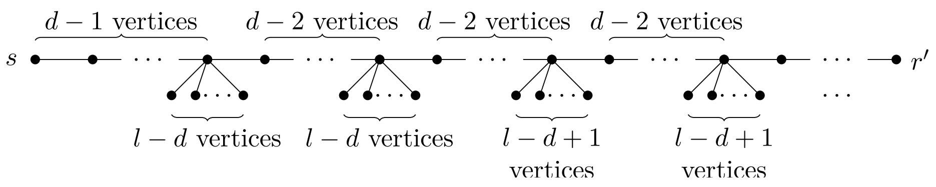


Fig. 4.3: The "squeezed" path in the proof of Thm 7

The NPM-instance is constructed as follows. The first tree T_1 is composed of one path P with $l \cdot B \cdot n^2 - 1$ vertices, and a set $\mathcal{P}$ of $3n$ paths. Each path p_i in $\mathcal{P}$ corresponds to an element $a_i \in A$ and contains $2 \cdot l - d + 1 + a_i$ vertices. Each path in $\mathcal{P}$ is connected to P by adding an edge between one of its end-vertices to the end-vertex r of P. The second tree T_2 consists of one "squeezed" path P' with $l \cdot B \cdot n^2 - 1$ vertices. To one end-vertex r' of P', we add $3 \cdot n \cdot (l - d + 2)$ new vertices as neighbors, called level-1 vertices. Every $(3 \cdot (l - d + 2))$-th level-1 vertex v has additional $3 \cdot (l - 1) - (d - 2)$ new leaves as neighbors and further, v is adjacent to one end-vertex of a path P_v that contains $d - 2$ vertices. The other end-vertex of P_v has additional B new leaves as neighbors. We refer to the leaf neighbors of v and the vertices in P_v as level-2 vertices.

Assume that there is a neighborhood-preserving mapping f from T_1 to T_2. Then, the vertices of P have to be mapped to the vertices of P' with $f(r) = r'$. Further, for each path p in $\mathcal{P}$, its first $l - d + 2$ vertices, ordered according to their distances to r, have to be mapped to the level-1 vertices. Then, the $(l - d + 3)$-th to the $(2 \cdot l - d + 1)$-th vertices of p have to be mapped to the level-2 vertices. Thus, a neighborhood-preserving mapping exists, if and only if there is a 3-partition of A. $\square$

4.1.2 Polynomial-Time Solvable Cases

In this subsection we present polynomial-time solvable cases of NPM on trees with $k = 0$. With $k = 0$ and $l = d = 1$, NPM on trees can be shown equivalent to the classical tree isomorphism problem and it is known that the isomorphism problem of trees can be solved in polynomial time [59].

Theorem 8. *NPM on trees can be solved in polynomial time for the case $k = 0$ and $l = d = 1$.*

Proof. We show the equivalence of the case of NPM to the tree isomorphism problem. Let T_1 and T_2 be trees. If T_1 is isomorphic to T_2, then there exists a mapping such that every edge $(v_1, v_2) \in E(T_1)$ iff $(f(v_1), f(v_2)) \in E(T_2)$. Thus, for every vertex v_1 and every its neighbor $v_2 \in N(v)$, $\text{dist}_{T_2}(f(v_1), f(v_2)) = 1$, which satisfies the condition of NPM with $l = 1$, $d = 1$, $k = 0$.

Now assume that there is a one-to-one mapping $f : V(T_1) \to V(T_2)$, such that $\forall\, v \in V(T_1)$, $\forall\, u \in N(v)$, $f(u) \in N(f(v))$, but the trees are not isomorphic. Then there exists $(u_1, u_2) \in E(T_2)$, such that $(f^{-1}(u_1), f^{-1}(u_2)) \notin E(T_1)$. This means that there exist $v_1, v_2 \in V(T_1)$, such that $f(v_1) = f(v_2)$, since $|E(T_1)| = |E(T_2)| = |V(T_1)| - 1$. This contradicts the definition of mapping f. □

Theorem 9. *NPM on trees can be solved in polynomial time for the case $k = 0$ and $l > d = 1$.*

Proof. We prove that a neighborhood-preserving mapping exists, if and only if $|V_1| \leq 2$. The "if"-direction is trivial. Suppose $|V_1| \geq 3$. Consider $u, v \in V_1$ with $(u, v) \in E_1$ and $x \in N^{\ell}_{T_1}(u)$, where $2 \leq \ell \leq l - 1$. If there exists a neighborhood-preserving mapping f, then $(f(u), f(v)) \in E_2$, $(f(u), f(x)) \in E_2$, and $(f(v), f(x)) \in E_2$, contradicting to the tree structure of T_2. □

Case $l = 2,\ d = 2,\ k = 0$

Next we present an algorithm for the case of NPM on trees when $l = 2$, $d = 2$, $k = 0$. We first present some conditions for vertices, under which a neighborhood-preserving mapping can exist. Assume that $|V_1| = |V_2| \geq 3$. We let leaves(T) denote the set of leaves of the tree T.

In the following we consider only the case where the diameter of T_2 is at least 4. If T_2 has a diameter of 2, then T_2 is a star and thus, an arbitrary mapping from T_1 to T_2 is a solution. For the case that T_2 has a diameter 3, there is a path in T_2 with 4 vertices a, b, c, d. Clearly, all other vertices are leaves adjacent to b or c. Observe that the diameter of T_1 should be at least 3, since otherwise, the given instance has no solution. Moreover, we cannot map two non-adjacent vertices u and v to b and c, since, otherwise, the neighborhoods of the vertices on the path between u and v cannot be preserved. Further, we cannot map a leaf u of T_1 to b or c, since otherwise, say mapping u to b, the whole T_1 has to be mapped to the star centered at b. Thus, if there exists a neighborhood-preserving mapping f, then we have two adjacent, non-leaf vertices v, u with $f(v) = b$ and $f(u) = c$. Clearly, there cannot be two neighbors of v such that one is mapped to a leaf adjacent to b and one to a leaf adjacent to c. So they are either all mapped to the leaves adjacent to b or all mapped to the leaves adjacent to c. Then, we can simply compare the numbers of leaves adjacent to b and to c with $|T_1(u)|$ and $|T_1(v)|$ to decide whether a neighborhood-preserving mapping exists. Here, $T_1(u)$ and $T_1(v)$ denote the subtrees, that result by deleting (u, v) from T_1 and contain u and v, respectively. This is clearly doable in polynomial time.

Some observations. In the following we present some observations which are crucial for the correctness of the algorithm.

Lemma 10. *Let $u, v \in V_1$ with $(u, v) \in E_1$. Suppose that there is a neighborhood-preserving mapping f with $(f(u), f(v)) \notin E_2$. Let a be the vertex in T_2 with $(f(u), a) \in E_2$ and $(f(v), a) \in E_2$. Then, it holds that for every vertex $z \in N_{T_1}(u) \cup N_{T_1}(v)$, $f(z) \in N_{T_2}[a]$.*

Proof. Clearly, for every vertex $z \in N_{T_1}(u) \cup N_{T_1}(v)$, we have $z \in N^2_{T_1}(u)$ and $z \in N^2_{T_1}(v)$. Thus, it must hold that $f(z) \in N^2_{T_2}(f(u))$ and $f(z) \in N^2_{T_2}(f(v))$. The claim follows from $N^2_{T_2}(f(u)) \cap N^2_{T_2}(f(v)) = N_{T_2}[a]$. □

Lemma 11. *Let $u, v \in V_1$ with $(u, v) \in E_1$. Suppose that the diameter of T_2 is at least 4. Then, a neighborhood-preserving mapping f with $(f(u), f(v)) \notin E_2$ exists, if and only if both $f(u)$ and $f(v)$ are in* leaves(T_2).

Proof. Clearly, if $f(u), f(v) \in \text{leaves}(T_2)$, then $(f(u), f(v)) \notin E_2$. Assume that the "only if"-direction is not true. Then, there is a neighborhood-preserving mapping f such that $(f(u), f(v)) \notin E_2$ and one of $f(u)$ and $f(v)$ is not in leaves(T_2), say $f(u) \notin \text{leaves}(T_2)$. Let a be the vertex in T_2 with $(f(u), a) \in E_2$ and $(f(v), a) \in E_2$. By Lemma 10, all vertices in $N_{T_1}(u) \cup N_{T_1}(v)$ are mapped to the vertices in $N_{T_2}[a]$. Root T_2 at a and denote the subtrees rooted at $f(u)$ and $f(v)$ by $T_2(f(u))$ and $T_2(f(v))$, respectively. Moreover, set $T_2(a) := T_2 \setminus (T_2(f(u)) \cup T_2(f(v)))$. By the assumption, $V(T_2(f(u))) \setminus \{f(u)\} \neq \emptyset$. However, since all vertices in $(N_{T_1}(u) \cup N_{T_1}(v)) \setminus \{u, v\}$ are mapped to the vertices in $T_2(a)$, there must be a vertex $b \in V(T_2(f(u))) \setminus \{f(u)\}$ such that b is mapped to a vertex x with $(x, y) \in E_1$ for a vertex $y \in V_1$, which is mapped to a vertex in $T_2(a)$. Obviously, $f(y) = a$ and y is in $N_{T_1}(u) \setminus \{v\}$. Thus, $V(T_2(f(v))) = \{f(v)\}$.

Root T_1 at u and let $T_1(y)$ be the subtree rooted at y. Then, all vertices $(N_{T_1}(u) \cup N_{T_1}(v)) \setminus \{y\}$ are mapped to $N_{T_2}(a)$. This implies, by Lemma 10, that for every vertex $z \in (N_{T_1}(u) \cup N_{T_1}(v)) \setminus \{u, v, y\}$, all vertices in $N_{T_1}(z)$ are mapped to $N_{T_2}(a)$. With the same argument, all vertices in $T_1 \setminus T_1(y)$ are mapped to $N_{T_2}(a)$. Finally, consider $T_1(y)$. Again, by Lemma 10, all vertices in $N_{T_1}(x) \cup N_{T_1}(y)$ are mapped to $N_{T_2}(f(u))$. The same holds also for other vertices in $T_1(y)$. Therefore, T_2 contains only two non-leaves, that is, vertices $f(u)$ and a, and has diameter 3, contradicting the assumption of the lemma. □

Lemma 12. *Let v be a leaf of T_1 with u being its only neighbor. Suppose that the diameter of T_2 is at least 4. If there is a neighborhood-preserving mapping f with $f(u) \notin$* leaves(T_2), *then $f(v) \in$* leaves(T_2).

Proof. Assume that the claim is not true. Let f be a mapping with $f(v), f(u) \notin \text{leaves}(T_2)$. Then, according to Lemma 11, $(f(u), f(v)) \in E_2$.

Let $T_2(f(u))$ and $T_2(f(v))$ be the subtrees which result by deleting $(f(u), f(v))$ from T_2 and contain $f(u)$ and $f(v)$, respectively. Let $f^{-1}(V(T_2(f(u))))$ and $f^{-1}(V(T_2(f(v))))$ be the sets of vertices in T_1 which are mapped to $T_2(f(u))$

and $T_2(f(v))$, respectively. Clearly, $f^{-1}(V(T_2(f(v)))) \setminus \{v\} \neq \emptyset$. Therefore, there must be a vertex $x \in f^{-1}(V(T_2(f(v)))) \setminus \{v\}$ which is adjacent to some vertex $y \in f^{-1}(V(T_2(f(u))))$. Obviously, $y = u$. This implies that $(u,x) \in E_1$ but $(f(u), f(x)) \notin E_2$. By Lemma 11, we have then $f(u), f(x) \in \text{leaves}(T_2)$, a contradiction. □

Lemma 13. *Suppose that the diameter of T_2 is at least 4. If a neighborhood-preserving mapping f exists with $f(u) \notin \text{leaves}(T_2)$ for $u \in V_1$, then for every $x \in N_{T_1}(u)$, we have $f(x) \in N_{T_2}(f(u))$.*

Proof. Consider an arbitrary vertex x in $N_{T_1}(u)$. Then, since $f(u)$ is not in leaves(T_2), according to Lemma 11, $(f(u), f(x)) \in E_2$ and thus, $f(x)$ has to be in $N_{T_2}(f(u))$. □

Lemma 14. *For $u, v \in V_1$ with $(u,v) \in E_1$, let $T_1(v)$ denote the subtree which results by deleting (u,v) from T_1 and contains v. Suppose that the diameter of T_2 is at least 4. If a neighborhood-preserving matching f exists with $f(u) \notin \text{leaves}(T_2)$ and $f(v) \in \text{leaves}(T_2)$, then for every vertex $x \in V(T_1(v))$, we have $f(x) \in \text{leaves}(T_2)$ and $(f(u), f(x)) \in E_2$.*

Proof. The claim is clearly true for v. By Lemma 11, $(f(u), f(v)) \in E_2$. Consider an arbitrary vertex $x \in (N_{T_1}(v) \setminus \{u\})$. Since $f(v) \in \text{leaves}(T_2)$, $f(x)$ has to be adjacent to $f(u)$. Moreover, since $(x,v) \in E_1$ and $(f(x), f(v)) \notin E_2$, $f(x)$ has to be a leaf of T_2. The same argument implies that all neighbors of x have to be leaves of T_2 adjacent to $f(u)$. □

If there is a mapping f which fulfills the condition of Lemma 14, that is, $f(u) \notin \text{leaves}(T_2)$ and $f(v) \in \text{leaves}(T_2)$ for two vertices $u, v \in V_1$ with $(u,v) \in E_1$, then we say that the subtree $T_1(v)$ is *absorbed* at $f(u)$. Clearly, subtree $T_1(v)$ cannot be absorbed, if the number of leaves adjacent to $f(u)$ is smaller than the number of vertices in $T_1(v)$. The following lemma summarizes the above observations.

Lemma 15. *Suppose T_2 has a diameter at least 4. Let $u, v \in T_1$ with $(u,v) \in E_1$. Let $T_1(u)$ and $T_1(v)$ be the subtrees resulting by deleting (u,v) from T_1 and containing u and v, respectively. Suppose that there exists a neighborhood-preserving mapping f with $(f(u), f(v)) \in E_2$. Let $T_2(f(u))$ and $T_2(f(v))$ be the subtrees resulting by deleting $(f(u), f(v))$ from T_2 and containing $f(u)$ and $f(v)$, respectively. Then, it holds that*

1. *either one of $f(u)$ and $f(v)$ (say $f(u)$) is a leaf of T_2 and for every vertices $x \in V(T_1(u))$, we have $f(x)$ being a leaf adjacent to $f(v)$,*
2. *or $|V(T_1(u))| = |V(T_2(f(u)))|$, $|V(T_1(v))| = |V(T_2(f(v)))|$, $|\text{leaves}(T_1(u))| \leq |\text{leaves}(T_2(f(u)))|$, and $|\text{leaves}(T_1(v))| \leq |\text{leaves}(T_2(f(v)))|$.*

Proof. Clearly, $f(u)$ and $f(v)$ cannot be both leaves. For the case that $f(u)$ is a leaf, the correctness follows directly from Lemma 14. For the case that both are not leaves, Lemma 13 implies that for all vertices $x \in N_{T_1}(u)$ (or $x \in N_{T_1}(v)$), we have $f(x) \in N_{T_2}(f(u))$ (or $f(x) \in N_{T_2}(f(v))$). Further, by Lemma 12, if x is a leaf, then $f(x)$ is a leaf too. For a non-leaf x, by Lemma 13 $f(y) \in N_{T_2}(f(x))$ for all vertices $y \in N_{T_1}(x)$. Thus, we can conclude that in this case, $|V(T_1(u))| = |V(T_2(f(u)))|$, $|V(T_1(v))| = |V(T_2(f(v)))|$, $|\text{leaves}(T_1(u))| \leq |\text{leaves}(T_2(f(u)))|$, and $|\text{leaves}(T_1(v))| \leq |\text{leaves}(T_2(f(v)))|$. $\square$

The algorithm. To ease the presentation, we assume that both trees T_1 and T_2 are rooted at $\text{root}(T_1)$ and $\text{root}(T_2)$, respectively, and the mapping f sought for satisfies $f(\text{root}(T_1)) = \text{root}(T_2)$. Further we assume that $\text{root}(T_2)$ is not a leaf. For a vertex v of a rooted tree T, we use $T(v)$ to denote the subtree rooted at v. The *labels* of a vertex $v \in V_2$, denoted by $\text{labels}(v)$, is a set of vertices from V_1 that can potentially be mapped to v. Clearly, $\text{labels}(\text{root}(T_2)) = \{\text{root}(T_1)\}$. For $U' \subseteq V_1$, we define $\text{labels}(v, U') := \text{labels}(v) \cap U'$. *Discarding* a label $u \in \text{labels}(v)$ is denoted by $\text{labels}(v) := \text{labels}(v) \setminus \{u\}$. By $\text{labels}(V')$ with $V' \subseteq V_2$ we denote the set $\bigcup_{v \in V'} \text{labels}(v)$. The algorithm consists of two phases, the first phase top-down preparing the labels of all vertices of T_2 and the second phase constructing the mapping from the labels in a bottom-up manner.

Phase 1. Starting at $\text{root}(T_2)$ with $\text{labels}(\text{root}(T_2)) = \{\text{root}(T_1)\}$, the algorithm iterates over all non-leaf vertices in T_2 according to the breadth-first order, and builds label sets for the children of a vertex $v \in V_2$ from the label set of v. Let $\text{ch}(u)$ denote the set of children of a vertex u in a rooted tree. For a vertex $v \in V_2$ and one of its labels $u \in \text{labels}(v)$, we process the children of u and v depending on their degrees as follows.

Leaf children of u. We first consider the leaf children of u. By Lemma 12, if v can be mapped to u, then all leaf children of u have to be mapped to the leaf children of v. Let l_u and l_v be the numbers of the leaf children of u and v, respectively. Thus, if $l_u > l_v$, then we discard u from $\text{labels}(v)$; otherwise, we select l_u many v's leaf children and store u's leaf children one-to-one in the label sets of the corresponding v's leaf children. We denote these l_u leaf children of v by $\mathcal{M}_{v,u}$.

Non-leaf children of v. For each non-leaf child v' of v, we iterate over all non-leaf children of u. If there is one non-leaf child u' of u satisfying $|V(T_1(u'))| = |V(T_2(v'))|$ and $|\text{leaves}(T_1(u'))| \leq |\text{leaves}(T_2(v'))|$, then we add u' to $\text{labels}(v')$; otherwise, we discard u from $\text{labels}(v)$. This is correct due to Lemma 15.

Now, $\text{labels}(v') \neq \emptyset$ for all non-leaf children $v' \in \text{ch}(v)$ and all leaf children of u are in the label sets of the leaf children in $\mathcal{M}_{v,u}$. The algorithm moves to the next vertex according the breath-first order.

Phase 2. In this phase, the algorithm processes the non-leaf vertices of T_2, in a reversed order of the first phase. For a vertex $v \in V_2$ and a label u from labels(v), it computes a maximum matching of the bipartite graph consisting of the non-leaf children of v and the non-leaf children of u. There is an edge between a non-leaf child v' of v and a non-leaf child u' of u, if and only if $u' \in$ labels(v'). If the matching is not perfect for the non-leaf children of v, then discard u from labels(v); otherwise, consider the non-leaf children of u which are not in the matching. By Lemma 14, all subtrees rooted at these non-leaf children of u have to be absorbed, that is, mapped to the leaf children of v, excluding the leaf children in $\mathcal{M}_{v,u}$. Then, the algorithm compares the total size of the subtrees rooted at these non-leaf children of u and the number of the leaf children of v that are not in $\mathcal{M}_{v,u}$. If they are not equal, then discard u from labels(v). If afterwards labels(v) $= \emptyset$, then return "no"; otherwise move to the next vertex.

Finally, at the root of T_2, if we have labels(root(T_2)) $=$ {root(T_1)}, then we can answer "yes".

Theorem 16. *NPM on trees can be solved in $O(n^{4+\omega})$ time for the case $k = 0$ and $l = d = 2$, where $n = |V_1| = |V_2|$ and $\omega = 2.38$.*

Proof. The correctness of the algorithm follows from its description. Concerning its running time, we fix a non-leaf vertex of T_2 as its root. By Lemma 15, we can try all possible non-leaf vertices of T_1 to be the root. In both phases, we iterate over all vertices in T_2 and each vertex in T_2 can have at most $|V_1|$ labels. When moving bottom-up we need to compute maximum matchings in bipartite graphs, which can be done in $O((|V_1| + |V_2|)^\omega)$ time (where $\omega = 2.38$ arises from the exponent of the matrix multiplication algorithm) [57]. Therefore, we have a total running time of $O(|V_1| \cdot |V_2| \cdot (|V_1| + |V_2|)^{2+\omega})$, that is, $O(n^{4+\omega})$ with $n = |V_1| = |V_2|$. □

By combining the ideas for proving Theorems 9 and 16, we can show that a neighborhood-preserving mapping between trees exists for $k = 0$ and $l > d = 2$, if and only if $|V_1| \leq 3$ or T_2 is a star.

Theorem 17. *NPM on trees can be solved in polynomial time for the case $k = 0$ and $l > d = 2$.*

4.2 NPM on Trees with $k > 0$

We show next that NPM on trees with $k > 0$ is NP-complete for all values of l and d. Then, we give two polynomial-time algorithms solving the special case of NPM that $k > 0$, $l = d = 1$, and one input tree is a path.

4.2.1 Two Input Trees

The NP-hardness results for NPM on trees with $k = 0$ can be easily generalized for the case $k > 0$. In the following we focus on the cases where NPM on trees with $k = 0$ can be solved in polynomial time.

Theorem 18. *NPM on trees is NP-complete, even if $k = 1$ and $l = d = 1$.*

Proof. We reduce again from 3-Partition. Given an instance (A, B) of 3-Partition, in order to construct the first tree T_1, we first create one long path and $3n$ "element" paths. The long path consists of $4B + 1$ vertices, while every element path corresponds to one particular element a_i, that is, the i-th element path contains a_i vertices. Moreover, we connect all element paths to the long path by adding an edge between one end-vertex of each element path and an end-vertex r of the long path. The second tree T_2 has only one vertex q with degree more than 2, which is connected to $n+2$ paths. Among these paths, two consist of $2B$ vertices and each of the other n paths contains B vertices.

Since T_1 has a vertex of degree $3n+1$ and the maximal degree of T_2 is $n+2$, at least one vertex of T_1 has to be added to the isolation set D. With $|D| \leq k = 1$, we have to add vertex r in T_1 to D. Moreover, the long path of T_1 has to be mapped to the two length-$(2B - 1)$ paths and vertex q. Thus, the $3n$ element paths have to be mapped to the n length-$(B-1)$ paths, which implies that there is a 3-partition. □

Theorem 19. *NPM on trees is NP-hard for the case $k > 0$ and $l > d = 1$.*

Proof. We reduce again from 3-Partition. Suppose that the given instance (A, B) of 3-Partition satisfies that all elements of A are even.

The tree T_1 contains only one vertex c with degree greater than 2; the degree of c is equal to $3n + 2$. Each neighbor of c is an end-vertex of a path with $\lfloor l/2 \rfloor$ vertices. In one special path, to the other end-vertex t, if l is even, we add two leaves as neighbors; if l is odd, we add one leaf as a neighbor to the only neighbor of t. We call the so far resulting tree a "spider".

To the end-vertices of the remaining paths, we attach the following $3n + 1$ paths. One path is a long path consisting of $4B + 2B(l - 1) + 1$ vertices. The others correspond to the elements in A, i.e., the i-th path consisting of $a_i + (l - 1) \cdot (a_i/2 - 1)$ vertices.

The tree T_2 has only one vertex c' with degree greater than 2; the degree of c' is equal to $(3n+2)\cdot\lfloor l/2\rfloor+2B(l-1)+\sum_{a_i\in A}(l-1)\cdot(a_i/2-1)+p+1$, where $p = 1$ if l is even, and $p = 0$, otherwise. Among these neighbors of c', n of them are the end-vertices of n paths, each of length $B-1$; one neighbor is an end-vertex of a path of length 1; two of the neighbors are the end-vertices of two paths, each with $2B$ vertices. Finally, we set $k := (3n+2)\cdot\lfloor l/2\rfloor+2B(l-1)+\sum_{a_i\in A}(l-1)\cdot(a_i/2-1)+p$ with $p = 1$ if l even, and $p = 0$, otherwise.

For the equivalence, observe that, from a set of vertices in T_1, that have pairwise distance at most l, at most two vertices can be in $V(T_1) \setminus D$; otherwise, we would need cycles in T_2. Thus, at most two vertices of the spider can be "kept", i.e., not in the isolation set. Further, for a path with $x + (l-1) \cdot x/2$ vertices with x being even, we need to delete at least $(l-1) \cdot x/2$ vertices to get a set of vertices such that no three vertices in this set have pairwise distance at most l. Then, we can conclude that, with k isolations allowed, if a mapping f exists, then, after deleting the isolation vertices, the remaining vertices must "induce with their l-neighborhoods" a set $\mathcal{P}$ of paths. Given a tree T and a set V' of vertices in T, the graph induced by V' with their l-neighborhoods has V' as its vertex set. There is an edge between $u, v \in V'$, if and only if the distance between u and v in T is at most l. In $\mathcal{P}$, there is a path with $4B+1$ vertices (remaining vertices of the long path), a length-1 path (two vertices kept in the spider), and $3n$ element paths, where the i-th path contains a_i vertices. Clearly, the length-$4B$ path has to be mapped to the two length-$2B$ paths. The two vertices kept in the spider are mapped the length-1 path attached to a neighbor of c'. The element paths can be mapped to the n paths of length $B-1$, if and only if the given 3-Partition instance is a yes-instance. □

Theorem 20. *NPM on trees is NP-complete, if $k > 0$ and $l \geq d = 2$.*

Proof. We provide a reduction from 3-Partition to NPM with $l = d = 2$. The same edge subdivision idea from the proof of Theorem 5 can be applied to prove the other cases.

The tree T_1 consists of a big star centered at a vertex c with $3n$ leaves, and $3n$ element stars. The i-th element star corresponds to a_i from A and has a_i leaves. Then, for each element star, an edge is added between its center and a distinct leaf of the big star. The tree T_2 consists of a path with $6n+1$ vertices. Suppose the vertices on this path are indexed $u_1, u_2, \ldots$ from the left to the right. We define $C := \{u_{2+3i} : 0 \leq i \leq n-1\}$. We add B leaves as neighbors to each of the vertices in C. Finally, we set $k := 3n+1$.

Clearly, if there is a 3-partition for (A, B), then we can isolate c and all its $3n$ neighbors and map the element stars according to the 3-partition to the stars centered at the vertices in C. For the other direction, observe that $|N^2_{T_1}(c)| = 6n+1$ and $|N^2_{T_1}(v)| > 3n$ for each neighbor v of c and there is no vertex u in T_2 with $|N^2_{T_2}(u)| \geq B+5$. Thus, one have to isolate c and all its neighbors. With $k = 3n+1$, the remaining $3n$ element stars have to be mapped to the stars centered at the vertices in C, corresponding to a 3-partition of A. □

4.2.2 $l = d = 1$, $k > 0$, and a Tree and a Path as Input

In contrast to the NP-hardness result of NPM on trees with $l = d = 1$ (Theorem 18), we show that if one of the input trees is a path, then NPM is polynomial-time solvable with $l = d = 1$.

The second tree is a path. To simplify the presentation, we reformulate NPM on trees with $l = d = 1$ and the second tree being a path as the following problem:

> Cutting Tree into Paths (CTP)
> **Input:** A tree T, an integer k.
> **Question:** Can we transform T to a set $\mathcal{P}$ of paths by deleting at most k vertices?

Lemma 21. *Given the second tree being a path, NPM on trees with $l = d = 1$ is equivalent to CTP.*

Proof. From a solution of CTP, we can simply set the isolation set D equal to the set of deleted vertices. Mapping the deleted vertices to the first $|D|$ vertices of P and then each path $p \in \mathcal{P}$ to the next $|V(p)|$ unmapped vertices on P clearly satisfies the neighborhood-preserving condition.

Let (T, P, k) be an instance of NPM and (D, f) be a solution of this instance with D being the isolation set and f the corresponding neighborhood-preserving mapping f from T to P. Since f is a neighborhood-preserving mapping and P has a maximal degree of 2, deleting the vertices in D from T clearly results in a set of paths. □

Next, we give a dynamic programming based algorithm solving CTP. Assume that T is rooted at an arbitrary vertex r, and let $T(v)$ denote the subtree rooted at a vertex v. Hereby, we distinguish at every vertex $v \in V(T)$ the following four cases:

1. v is deleted,
2. v is an end-vertex of a path in $\mathcal{P}$ and all children of v are deleted,
3. v is on a path in $\mathcal{P}$ and exactly one end-vertex of this path is in $V(T(v)) \setminus \{v\}$,
4. v is on a path in $\mathcal{P}$ and both end-vertices of this path are in $V(T(v)) \setminus \{v\}$.

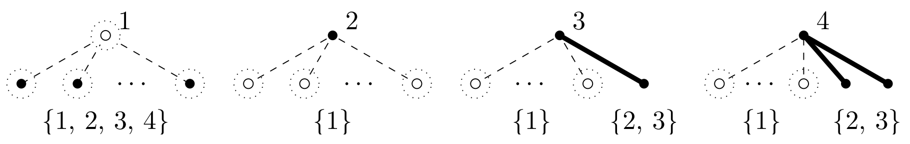


Fig. 4.4: The four cases for CTP when computing function d_v; doted circles denote that the vertex can be in Case 1

We define further a function $d_v(c)$ for v with $c \in \{1, 2, 3, 4\}$, denoting one of the previously defined cases. This function $d_v(c)$ stores the minimal number of deletions needed in $T(v)$ to derive a set of paths, where v follows Case c. We recursively compute $d_v(c)$ for all vertices $v \in V(T)$ and all four cases in a bottom-up way. Clearly, at the root r, if $\min_{c=1\ldots4} d_r(c) \leq k$, then we return "yes"; otherwise, return "no".

At a leaf vertex v, Cases 3 and 4 clearly cannot be applied. We can easily set $d_v(1) := 1$, $d_v(2) := 0$, and $d_v(3) = d_4(v) := \infty$. The computation of $d_v(c)$ for a non-leaf vertex v distinguishes again four cases (see Fig. 4.4 for a depiction of the cases):

1. $d_v(1) := \sum_{u \in \mathrm{ch}(v)} \min_{c=1\ldots4} d_u(c) + 1$,
2. $d_v(2) := \sum_{u \in \mathrm{ch}(v)} d_u(1)$,
3. $d_v(3) := \min_{u \in \mathrm{ch}(v)} (\sum_{u' \in (\mathrm{ch}(v) \setminus \{u\})} d_{u'}(1) + \min_{c=2,3} d_u(c))$,
4. $d_v(4) := \min_{u_1, u_2 \in \mathrm{ch}(v)} (\sum_{u \in (\mathrm{ch}(v) \setminus \{u_1, u_2\})} d_u(1) + \min_{c=2,3} d_{u_1}(c) + \min_{c=2,3} d_{u_2}(c))$.

Theorem 22. *CTP can be solved in $O(|V(T)|^3)$ time.*

Proof. The correctness of the algorithm follows from the definitions of the cases and $d_v(c)$ and the recursive computation of $d_v(c)$. Clearly, the most time-consuming computation at a vertex v is for $d_v(4)$. Here, one needs to consider every pair of the children of v. This results in a running time of $O(|V(T)|^3)$. □

Corollary 23. *NPM on trees with the second tree being a path can be solve in $O(|V(P)|^3)$ time for $l = d = 1$.*

The first tree is a path. Again, NPM on trees with $l = d = 1$ and the first tree being a path can be reformulated as the following problem:

Fitting Path to Tree by Deletions (FPTD)
Input: A path P and a tree T with $|V(P)| = |V(T)|$, an integer k.
Question: Can we delete at most k vertices from P such that there exists a subgraph T' of T isomorphic to the resulting set $\mathcal{P}$ of paths?

With $l = d = 1$, the following lemma is easy to prove.

Lemma 24. *If the first tree is a path, then NPM on trees with $l = d = 1$ is equivalent to FPTD.*

In the following, we give a polynomial-time algorithm solving FPTD. Again we assume that T is rooted at an arbitrary vertex r and denote by $T(v)$ the subtree rooted at a vertex v. Let D denote the set of the vertices whose deletion from P results in a set $\mathcal{P}$ of paths. We extend the isomorphic mapping f from $V(\mathcal{P})$ to $V(T')$ to a mapping from $V(P)$ to $V(T)$, by assigning an arbitrary one-to-one correspondence between D and $V(T) \setminus V(T')$. This is doable since $|V(P)| = |V(T)|$. The algorithm processes the vertices in T in a bottom-up manner. At each vertex v, it distinguishes the following 6 cases concerning the way how v is mapped by the mapping f:

1. v is mapped to a vertex in D;
2. v is mapped to a path $p \in \mathcal{P}$ with $|V(p)| = 1$;
3. v is mapped to an end-vertex of a path in $\mathcal{P}$, whose other end-vertex is mapped to a vertex not in $V(T(v))$;
4. v is mapped to an end-vertex of a path in $\mathcal{P}$, whose other end-vertex is mapped to a vertex in $V(T(v)) \setminus \{v\}$;
5. v is mapped to a non-end vertex of a path in $\mathcal{P}$, which has one end-vertex mapped to a vertex in $V(T(v)) \setminus \{v\}$ and another one mapped to a vertex not in $V(T(v))$;
6. v is mapped to a non-end vertex of a path in $\mathcal{P}$, whose both end-vertices are mapped to vertices in $V(T(v)) \setminus \{v\}$.

For each of the cases, the algorithm checks whether it is possible to delete some vertices to create a set of paths, which can be mapped to $T(v)$, given the mapping of v following this case. If so, it stores the minimum possible number of deletions. Notice that Case 1 causes additional caution in this check. On the one hand, the subtree $T(v)$ could be mapped to some set of paths, which however need to delete a lot of vertices from P. These deleted vertices might be mapped to vertices of Case 1, which are outside of $T(v)$. On the other hand, we might have a lot of vertices in $T(v)$ with Case 1. However, the paths mapped to $T(v)$ do not cause so many vertex deletions. Thus, we introduce an additional parameter s to record this information with $-k \leq s \leq k$. If we say that there are s "mappable" vertices in $T(v)$, we mean the following: If $s < 0$, there are $|s|$ vertices which are deleted to create the paths mapped to $T(v)$ but are not mapped to the vertices with Case 1 in $T(v)$; otherwise, there are s

vertices with Case 1 in $T(v)$, which can be mapped to vertices deleted to create paths mapped to vertices outside of $T(v)$. Thereupon, we define the dynamic programming table F_v at vertex v with two parameters, one representing the 6 cases and the other being s. The entry $F_v(c, s)$ contains the minimal number of vertex deletions needed to create a set of paths in P, which are mapped to $T(v)$, under the conditions that v follows Case c and there are s mappable vertices in $T(v)$.

To ease the presentation, we say to "open" a path at v, if v is in Cases 2 and 3. Notice that, once we open a path, we increase the number of vertex deletions by one. However, since by deleting i vertices from P we can create $i + 1$ paths, we check whether $F_r(c, -1) \leq k + 1$ for some Case c at the root r. If so, we return "yes"; otherwise, we return "no".

It remains to describe the computation of $F_v(c, s)$. At a leaf v, it is clear that only Cases 1, 2, and 3 can be applied. Thus, all entries of F_v are set to ∞, except for three entries, where we set $F_v(1, 1) := 0$, $F_v(2, -1) := 1$, and $F_v(3, -1) := 1$. The correctness here is obvious.

At a non-leaf vertex v, let $\mathrm{ch}(v) = \{u_1, \ldots, u_d\}$, where d is the number of children of v. We define three additional tables:

- For $-k \leq s \leq k$ and $1 \leq i \leq d$, $A_v(s, i)$ stores the minimal number of deletions needed to create a set of paths in P, which are mapped to the subtrees rooted at $u_1, \ldots, u_i$, under the conditions that $u_1, \ldots, u_i$ are of Case 1, 2, 4 or 6, and there are s mappable vertices in these subtrees;

- For $-k \leq s \leq k$, $1 \leq i \leq d$, and $1 \leq j \leq i$, $B_v(s, i, j)$ stores the minimal number of deletions needed to create a set of paths in P, which are mapped to the subtrees rooted at the vertices in $\{u_1, \ldots, u_i\} \setminus \{u_j\}$, under the conditions that all vertices in $\{u_1, \ldots, u_i\} \setminus \{u_j\}$ are of Case 1, 2, 4 or 6, and there are s mappable vertices in these subtrees;

- For $-k \leq s \leq k$, $1 \leq i \leq d$, and $1 \leq j_1, < j_2 \leq i$, $C_v(s, i, j_1, j_2)$ stores the minimal number of deletions needed to create a set of paths in P, which are mapped to the subtrees rooted at $\{u_1, \ldots, u_i\} \setminus \{u_{j_1}, u_{j_2}\}$, under the conditions that all vertices in $\{u_1, \ldots, u_i\} \setminus \{u_{j_1}, u_{j_2}\}$ are of Case 1, 2, 4 or 6, and there are s mappable vertices in these subtrees;

- For $-k \leq s \leq k$, $1 \leq i \leq d$, and $i < j \leq d$, $D_v(s, i, j)$ stores the minimal number of deletions needed to create a set of paths in P, which are mapped to the subtrees rooted at u_i and u_j, under the conditions that u_i and u_j are of Case 3 or 5, and there are s mappable vertices in these subtrees.

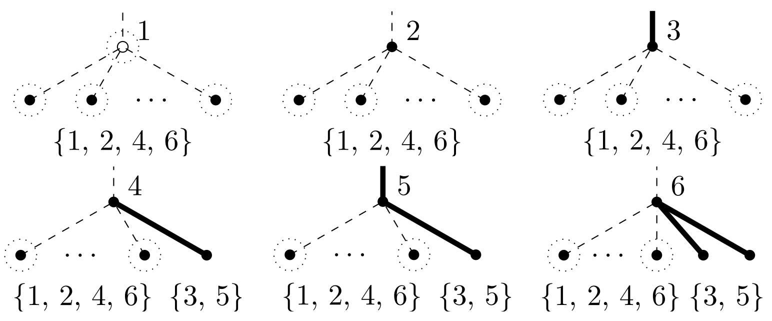


Fig. 4.5: The six cases for FPTD when computing table F_v; doted circles denote that the vertex can be in Case 1

To compute the three tables, we apply the following recursions: Initialize $A_v(s, 0) := 0$ for $-k \le s \le k$. For each $i = 1, \dots, d$ and each $s = -k, \dots, k$, set

$$A_v(s, i) := \min_{-k \le q \le k} (A_v(s - q, i - 1) + \min_{c \in \{1,2,4,6\}} F_{u_i}(c, q)) \ .$$

In order to fill in $B_v(s, i, j)$, initialize $B_v(s, i, i) := A_v(s, i - 1)$ for every $-k \le s \le k$ and $2 \le i \le d$. For $1 \le j < i$, the recursion for B_v is as follows:

$$B_v(s, i, j) := \min_{-k \le q \le k} (B_v(s - q, i - 1, j) + \min_{c \in \{1,2,4,6\}} F_{u_i}(c, q)) \ .$$

Then, initialize $C_v(s, i, j, i) := B_v(s, i, j)$ for every $-k \le s \le k$, $2 \le i \le d$, and $1 \le j < i$. For $1 \le j_1 < j_2 < i$, the recursion for C_v is as follows:

$$C_v(s, i, j_1, j_2) := \min_{-k \le q \le k} (C_v(s - q, i - 1, j_1, j_2) + \min_{c \in \{1,2,4,6\}} F_{u_i}(c, q)) \ .$$

Finally, for each $1 \le i < j \le d$ and $-k \le s \le k$, we compute D_v as follows:

$$D_v(s, i, j) := \min_{-k \le q \le k} (\min_{c \in \{3,5\}} F_{u_i}(c, q) + \min_{c \in \{3,5\}} F_{u_j}(c, s - q)) \ .$$

The correctness of the computation of the tables follows from the recursions. We compute F_v for each of the 6 cases as follows (see Fig. 4.5 for a depiction of the cases):

Case 1. The vertex v should be mapped to a deleted vertex. Then, the children of v should be of Cases 1, 2, 4, and 6. We need only to sum up the deletions needed to create the paths mapped to the subtrees rooted at the children. Notice that we have one additional vertex v of Case 1 which is not mapped. We set $F_v(1, s) := A_v(s - 1, d)$.

Case 2. We have to open a new path p with $|V(p)| = 1$ mapped to v. This implies that we need one more vertex deleted in $T(v)$ than in the forest consisting

of the subtrees rooted at the children of v. The cases for the children are the same as in Case 1. Thus, we set $F_v(2,s) := A_v(s+1,d)+1$.
Case 3. As in Case 2, we open a new path p at v. Therefore, $F_v(3,s) := F_v(2,s)$.
Case 4. One path should end at v and has its other end-vertex in $V(T(v))\setminus\{v\}$. Thus, at least one of v's children has to be of Case 3 or 5, while the other are of Cases 1, 2, 4, and 6. We set

$$F_v(4,s) := \min_{u_i \in \mathrm{ch}(v), -k \le q \le k} \left(B_v(s-q,d,i) + \min_{c\in\{3,5\}} F_{u_i}(c,q)\right) \ .$$

Case 5. The vertex v is mapped to a non-end vertex of a path with one end-vertex mapped inside of $T(v)$ and the other outside of $T(v)$. Therefore, one child of v must be of Case 3 or 5, while the others are of Cases 1, 2, 4, and 6. We have the same situation as Case 4: $F_v(5,s) := F_v(4,s)$.
Case 6. With both end-vertices mapped inside of $T(v)$, two children of v must be of Cases 3 and 5. Note that with two paths "merging" at v, we have in $T(v)$ one path less than in the forest consisting of the subtrees rooted at the children of v. With C_v and D_v, we compute $F_v(6,s)$ as follows:

$$F_v(6,s) := \min_{u_i,u_j \in \mathrm{ch}(v), -k \le q \le k} \left(C_v(s+1-q,d,i,j) + D_v(q,i,j)\right) - 1 \ .$$

At the root r, if $\min_{c\in\{1,2,4,6\}} F_r(c,-1) \le k+1$, then we return "yes"; otherwise, "no".

Theorem 25. *FPTD can be solved in $O(|V(T)|^4 \cdot k^2)$ time.*

Proof. The correctness of the algorithm follows from the definition of the recursions. Concerning the running time, it is not difficult to see that the most time-consuming computation at a vertex v is for the table $C_v(s,i,j_1,j_2)$. Since the table has size of $O(k \cdot |V(T)|^3)$ and each entry of C_v causes an iteration over q, the computation of C_v needs $O(k^2 \cdot |V(T)|^3)$ time. Thus, the overall running time is $O(|V(T)|^4 \cdot k^2)$. □

Corollary 26. *NPM on trees with the first tree being a path can be solved in $O(|V(T)|^4 \cdot k^2)$ time.*

4.3 Integer Linear Program Formulation for NPM

The optimization version of NPM asks for the minimal number of isolated vertices (k). In the presented ILP formulation of the problem we seek for a mapping between vertices of trees. In the mapping isolated vertices are not mapped to any other vertices (remember that an isolated vertex can cover any vertex in a graph). Instead of minimizing the number of isolated vertices, we maximize the number of mapped vertices. Then, NPM can be presented as follows:

$$\text{maximize} \sum_{\substack{t \in V(T_1),\\ w \in V(T_2)}} x_{tw} \ , \tag{4.1}$$

$$\begin{aligned}
\text{subject to} \qquad & x_{tw} = \{0,1\} \ , && \forall\, t \in V(T_1),\ \forall\, w \in V(T_2) \\
& \sum_{w \in V(T_2)} x_{tw} \le 1 \ , && \forall\, t \in V(T_1) \qquad (4.2) \\
& \sum_{t \in V(T_1)} x_{tw} \le 1 \ , && \forall\, w \in V(T_2) \qquad (4.3)
\end{aligned}$$

$$\begin{aligned}
& \sum_{w \in V(T_2)} x_{tw} + \sum_{w \in V(T_2)} x_{uw} + x_{tv} - \sum_{w \in N_d(v)} x_{uw} \\
& \qquad + \Big(\sum_{\substack{u' \in V(\text{path}_{T_1}(t,u)),\\ w \in V(T_2)}} x_{u'w} - |\text{path}_{T_1}(t,u)| \Big) < 3 \ , \qquad (4.4) \\
& \qquad\qquad \forall\, t \in V(T_1), \forall\, u \in N_l(t), \forall\, v \in V(T_2)
\end{aligned}$$

Theorem 27. *The above ILP formulation of the optimization version of NPM is correct.*

Proof. We set x_{tw} to 1 whenever vertex $t \in V(T_1)$ is matched to vertex $w \in V(T_2)$; otherwise x_{tw} is 0. Conditions (4.2) and (4.3) ensure that every vertex in T_1 is mapped to at most one vertex in T_2 and vice versa. In condition (4.4) the first sum is equal to 1 if vertex t, was mapped (and not deleted); the second sum is 1 if u, an l-neighbor of t was mapped (and not deleted). The third term, x_{tv}, is 1 if vertex t was mapped to vertex v, and 0 otherwise. The third sum (the forth term) equals to 1 if u was mapped within d-neighborhood of v, and 0

Tab. 4.2: All cases for the first four terms in the left part of condition (4.4) in ILP formulation of NPM

Case:	1	2-8		9	10	11	12	13	14	15	16
$\sum_{w \in V(T_2)} x_{tw}$	0	0	$\{0,1\}$	0	0	1	1	1	1	1	1
$\sum_{w \in V(T_2)} x_{uw}$	0	$\{0,1\}$	0	1	1	0	0	1	1	1	1
x_{tv}	0	1	$\{0,1\}$	0	0	0	1	0	0	1	1
$-\sum_{w \in N_d(v)} x_{uw}$	0	$\{0,-1\}$	-1	0	-1	0	0	0	-1	0	-1
Value:	0	-		1	0	1	2	2	1	3	2

otherwise. The expression in braces is 0 if no vertex on the path between t and u is isolated, it is negative otherwise.

To ease the proof, in Table 4.2 we present all possible combinations of values for the first four terms in the left part of condition (4.4).

As it can be seen from the table, in Case 15 vertex t and its l-neighbor u are mapped, but beyond the distance d, which is acceptable if there is an isolated vertex on the path between t and u (then the left part of condition (4.4) is less than 3). If no vertex is isolated on the path, then the value of the term in braces in condition (4.4) equals to 0, and the left part of the condition is equal to 3. This means that, under given conditions, Case 15 is the only one that does not satisfy condition (4.4). □

The weighted version of the problem should maximize the similarity between matched pairs of vertices, as well as the "importance" of matched vertices.

5 Compactness-Preserving Mapping on Trees

In this chapter we introduce and study a new generalization of the graph isomorphism problem. We call this problem Compactness-Preserving Mapping (CPM). Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, CPM asks if there is a one-to-one mapping $f : V_1 \to V_2$, such that for every vertex v from G_1 the vertices that are "close" to v in G_1 are mapped to the vertices in G_2 that are close in total to $f(v)$. Formally, the problem is defined as follows. Here $N^i_G(v)$ denotes the *i-neighborhood* of v, i.e., the set of vertices that have distance at most i to v in the graph G. L_v and L'_v are defined as follows. Given a graph G and an integer l, the *proper distance* L_v of vertex $v \in V(G)$ is defined as $L_v := \sum_{u \in N^l_G(v)} \operatorname{dist}_G(v, u)$. Given an integer l, two graphs G_1 and G_2 and a mapping $f : V(G_1) \to V(G_2)$, the *image distance* L'_v of vertex $v \in V(G_1)$ is defined as $L'_v := \sum_{u \in N^l_{G_1}(v)} \operatorname{dist}_{G_2}(f(v), f(u))$. The formal definition of CPM is as follows:

> Compactness-Preserving Mapping (CPM)
> **Input:** Graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $|V_1| = |V_2|$, non-negative integers l and d.
> **Question:** Is there a one-to-one mapping $f : V_1 \to V_2$ such that $\forall v \in V_1$, $L'_v - L_v \leq d$?

We call a solution mapping f of CPM *compactness-preserving.*

The study of CPM is mainly motivated by the protein-protein interaction (PPI) network alignment problem. The input of the problem consists of two graphs, i.e., PPI networks of two species, where vertices represent proteins and edges correspond to interactions between pairs of vertices. The output is a mapping between input graphs, which ideally maps proteins (i.e. vertices) of the same biological function and/or of the same evolutionary origin. We propose CPM to model the network alignment problem for the following reasons. Since throughout the evolution many interactions (i.e. edges of the graphs)

Tab. 5.1: Summary of the cases for CPM on trees (P stands for polynomial, NPC for NP-complete)

	$d = 0$	$d = 1$	$d \geq 2$
$l = 1$	P (Thm. 32)	P (Thm. 44)	NPC (Thm. 29)
$l = 2$	P (Thm. 38)	P (Thm. 54)	NPC (Thm. 30)
$l \geq 3$	NPC (Thm. 31)	NPC (Thm. 31)	NPC (Thm. 31)

remain conserved, that, as consequence, results in mostly conserved neighborhoods of the vertices, a biologically meaningful mapping between two PPI networks should also map larger part of the corresponding interactions. However, the requirement that all edges have to be mapped exactly is too strict, since not all interactions are conserved and the data is noisy and incomplete. The parameter l of CPM determines the radius of the conserved neighborhoods of the vertices, while the parameter d can be used to reflect the degree of noise and incompleteness of data. Thus, given two PPI networks as input a mapping of CPM with large value of l and small value of d could represent a biologically meaningful alignment. The similar can be applied to the backbone subtrees extracted from PPI networks.

Note that CPM can easily be generalized to take into account an isolation set in the similar way as in Neighborhood-Preserving Mapping (NPM). As for NPM, the approximate version of CPM on trees can be shown to be NP-complete for all values of l and d by a reduction from the subforest isomorphism problem [20].

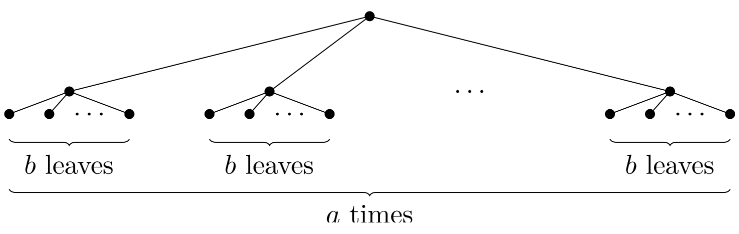
In this chapter we study the classical complexity of CPM with input graphs being trees with respect to different values of parameters l and d. The findings are summarized in Table 5.1: CPM on trees is polynomial-time solvable only if $l \leq 2$ and $d \leq 1$; all other cases turn out to be NP-complete. First, we provide proofs for the NP-hard cases. Next, we investigate polynomial-time solvable cases. At the end of the chapter we provide an integer linear program formulation of an optimization version of CPM with isolation set.

5.1 NP-Hardness Results

We first present the proofs of the NP-hard cases. Note that, given a graph G, for $v \in V(G)$ we have $L_v := \sum_{u \in N^l_{G_1}(v)} \mathrm{dist}_G(v, u) = \sum_{i=1}^{l} i \cdot |\hat{N}^i_G(v)|$.

5.1.1 Case $l = 1$, $d \geq 2$

The correctness of the following lemma is not difficult to prove:



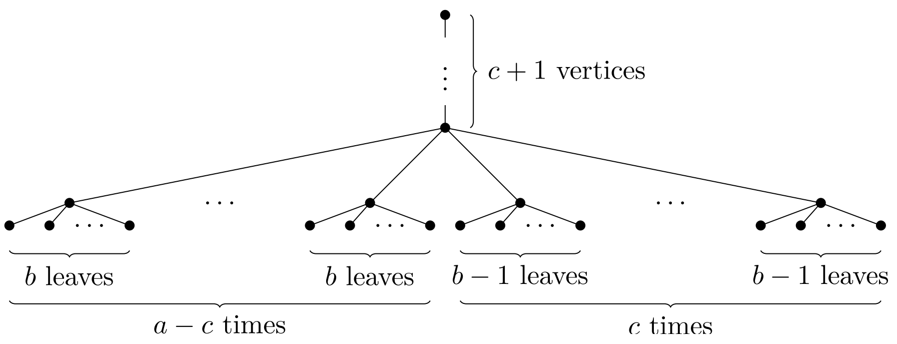


Fig. 5.1: A donor tree $\mathrm{DT}_1(a,b)$ and a recipient tree $\mathrm{RT}_1(a,b,c)$ defined in the proof of Thm. 29

Lemma 28. *If there is a compactness-preserving mapping f from T_1 to T_2 with $l=1$, then for any $v \in V_1$ it holds $|N_{T_2}(f(v))| + d \geq |N_{T_1}(v)|$.*

Proof. Let $N_1 := |N_{T_1}(v)|$ and $N_2 := |N_{T_2}(f(v))|$. Suppose the lemma is not true, i.e. $N_2 + d < N_1$. Then, there are at least $(N_1 - N_2)$ neighbors of v that are mapped at distance at least two from $f(v)$ in T_2. Then, $L'_v - L_v \geq (N_2 + 2 \cdot (N_1 - N_2)) - N_1 = N_1 - N_2 > (N_2 + d) - N_2 > d$, which is a contradiction. □

Theorem 29. *CPM on trees with $l = 1$ and $d \geq 2$ is NP-complete.*

Proof. Clearly, given a mapping from T_1 to T_2, it can be tested in polynomial time whether it is compactness-preserving.

In order to show the NP-hardness we provide a reduction from 3-Partition.

In the following, given an instance (A, B) of 3-Partition with all elements of A being even, we construct the corresponding instance $(T_1, T_2, l = 1, d \geq 2)$ of CPM on trees.

We first introduce two graph gadgets. A *donor tree* $\rho := \mathrm{DT}_1(a,b)$ consists of one vertex, called the *center vertex* of ρ, and a stars, each having b leaves (see Fig. 5.1). The centers of the stars are made adjacent to the center vertex of ρ. A *recipient tree* $\tilde{\rho} := \mathrm{RT}_1(a,b,c)$ consists of a path with $c+1$ vertices, $a-c$ stars, each having b leaves, and c stars, each having $b-1$ leaves (see Fig. 5.1).

The centers of all stars are connected to an end-vertex v of the path. We call v the *center vertex* of $\tilde{\rho}$, and the other end-vertex of the path the *tail* of $\tilde{\rho}$. Also note that the number of vertices in a recipient tree $\mathrm{RT}_1(a,b,c)$ is independent of c and equal to $|\mathrm{DT}_1(a,b)| = a \cdot (b+1) + 1$.

The tree T_1 consists of the following components:

- one star S_0 with the center c and $3n$ leaves,
- one "big" donor tree $D_\mathrm{b} := \mathrm{DT}_1(N_2, N_1)$,
- a set $\mathcal{D}_\mathrm{m}$ of $3n$ "middle" donor trees $\mathrm{DT}_1(N_4, N_3)$,
- a set $\mathcal{D}_\mathrm{s}$ of nB "small" donor trees $\mathrm{DT}_1(N_6, N_5)$, and
- a set $\mathcal{P}$ of $3n$ paths; the i-th path for $i = 1, \ldots, 3n$ corresponds to one particular element a_i of A and has a_i vertices.

Here $N_i := Bdn^{8-i}$, for $i = 1, \ldots, 6$.

The components are connected to T_1 as follows. Every vertex of a path from $\mathcal{P}$ is connected to the center vertex of one particular donor tree from $\mathcal{D}_\mathrm{s}$, as depicted in Fig. 5.2. Then, for every leaf v of S_0, we connect to v a middle donor tree $\rho \in \mathcal{D}_\mathrm{m}$ by adding an edge between v and the center of ρ. We also add en edge between v and one end-vertex of a path in $\mathcal{P}$. Finally, the big donor tree is connected to S_0 by adding an edge between its center vertex and c.

The tree T_2 consists of the following components:

- one star $\tilde{S}_0$ with the center $\tilde{c}$ and $3n$ leaves,
- one "big" recipient tree $\tilde{R}_\mathrm{b} := \mathrm{RT}_1(N_2, N_1, d+1)$,
- a set $\tilde{\mathcal{R}}_{\mathrm{m}_1}$ of n "middle" recipient trees $\mathrm{RT}_1(N_4, N_3, d+1)$,
- a set $\tilde{\mathcal{R}}_{\mathrm{m}_2}$ of $2n$ "middle" recipient trees $\mathrm{RT}_1(N_4, N_3, d-1)$,
- a set $\tilde{\mathcal{R}}_\mathrm{s}$ of nB "small" recipient trees $\mathrm{RT}_1(N_6, N_5, d-1)$, and
- a set $\tilde{\mathcal{P}} := \bigcup_{j=1}^{n} \tilde{\mathcal{P}}_j$, where $\tilde{\mathcal{P}}_j$ is a set of length-1 paths with $|\tilde{\mathcal{P}}_j| = B/2$.

The components are combined to T_2 as shown in Fig. 5.2. Every vertex $v \in V(\tilde{\mathcal{P}})$ is connected to one particular recipient tree $\tilde{\rho}$ from $\tilde{\mathcal{R}}_\mathrm{s}$ by adding an edge between v and the tail of $\tilde{\rho}$. Note that the distance between v and the center vertex of $\tilde{\rho}$ is equal to $d-1$. The vertex $\tilde{c}$ is made adjacent to the tail of $\tilde{R}_\mathrm{b}$; every leaf of $\tilde{S}_0$ is made adjacent to the tail of one particular tree from $\tilde{\mathcal{R}}_{\mathrm{m}_1} \cup \tilde{\mathcal{R}}_{\mathrm{m}_2}$. By $\mathcal{L}_{m_1}$ and $\mathcal{L}_{m_2}$ we denote the sets of the leaves of $\tilde{S}_0$ that are adjacent to the recipient trees of $\tilde{\mathcal{R}}_{\mathrm{m}_1}$ and $\tilde{\mathcal{R}}_{\mathrm{m}_2}$, respectively. Note that the distance between the center vertex of a recipient tree $\tilde{\rho} \in \tilde{\mathcal{R}}_{m_1}$ and the corresponding leaf in $\mathcal{L}_{m_1}$ is $d+1$, which is equal to the distance between the tail of $\tilde{\rho}$ and the centers of the stars of $\tilde{\rho}$. Similarly, the distance between the center vertex of a recipient tree $\tilde{\rho} \in \tilde{\mathcal{R}}_{m_2}$ and the corresponding leaf in $\mathcal{L}_{m_2}$ is equal to $d-1$. Finally, we add an edge between the j-th vertex in $\mathcal{L}_{m_1}$ and one end-vertex of every path from $\tilde{\mathcal{P}}_j$, $j = 1, \ldots, n$.

Obviously, the time needed to construct T_1 and T_2 is polynomial in n, B, and d.

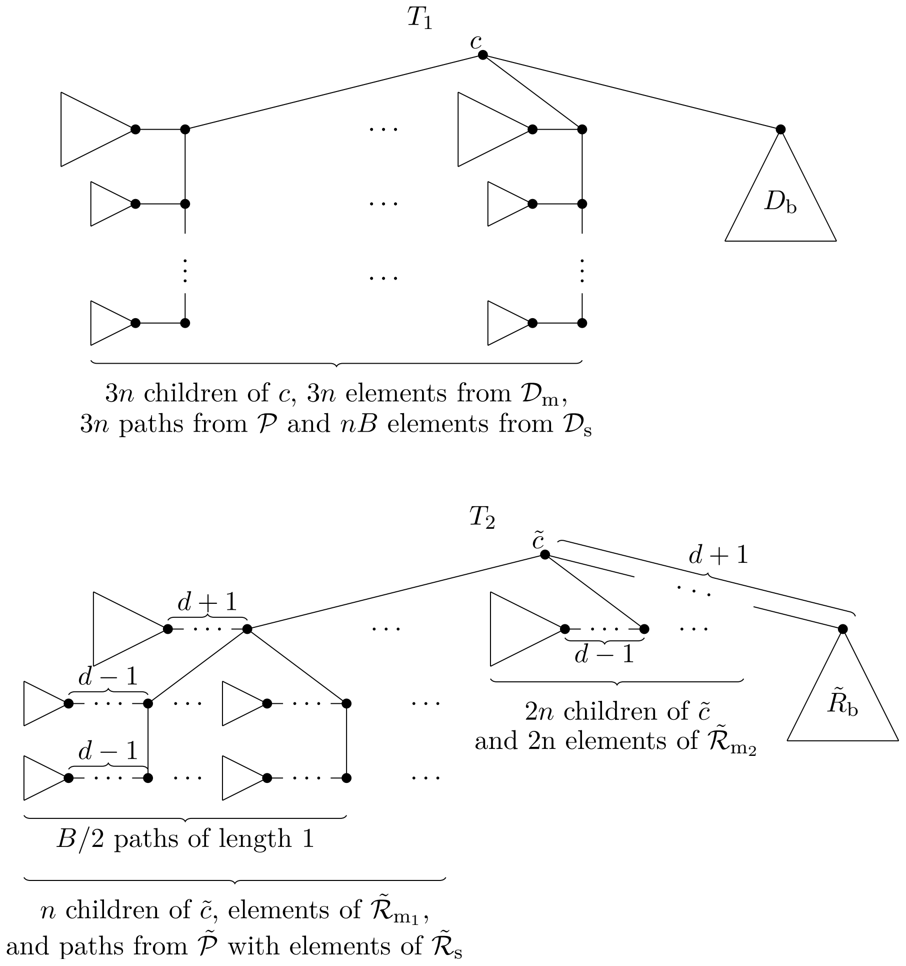

T_1
c
D_{b}
$3n$ children of c, $3n$ elements from $\mathcal{D}_{\mathrm{m}}$,
$3n$ paths from $\mathcal{P}$ and nB elements from $\mathcal{D}_{\mathrm{s}}$
T_2
$\tilde{c}$
$d+1$
$d+1$
$d-1$
$d-1$
$d-1$
$2n$ children of $\tilde{c}$
and 2n elements of $\tilde{\mathcal{R}}_{\mathrm{m}_2}$
$\tilde{R}_{\mathrm{b}}$
$B/2$ paths of length 1
n children of $\tilde{c}$, elements of $\tilde{\mathcal{R}}_{\mathrm{m}_1}$,
and paths from $\tilde{\mathcal{P}}$ with elements of $\tilde{\mathcal{R}}_{\mathrm{s}}$

Fig. 5.2: The trees T_1 and T_2 of CPM on trees corresponding to an instance of 3-PARTITION as defined in the proof of Thm. 29

"$\Longrightarrow$": Let $A_1, \dots, A_n$ be a 3-partition for an instance (A, B) of 3-PARTITION. The mapping f is constructed as follows. The vertices in S_0 are mapped to $\tilde{S}_0$ with $\tilde{c} = f(c)$, the vertices in D_{b} to $\tilde{R}_{\mathrm{b}}$, the donor trees in $\mathcal{D}_{\mathrm{m}}$ to the recipient trees in $\tilde{\mathcal{R}}_{\mathrm{m}_1}$ and $\tilde{\mathcal{R}}_{\mathrm{m}_2}$, the donor trees in $\mathcal{D}_{\mathrm{s}}$ to the recipient trees in $\tilde{\mathcal{R}}_{\mathrm{s}}$. Concerning the mapping from a donor tree ρ to the corresponding recipient tree $\tilde{\rho}$, the centers of the stars in ρ are mapped to the centers of the stars in $\tilde{\rho}$ and the center vertex of ρ is mapped to the center vertex of $\tilde{\rho}$. For $\tilde{\rho} \in \tilde{\mathcal{R}}_{\mathrm{m}_1}$ (or $\tilde{\rho} \in \tilde{\mathcal{R}}_{\mathrm{m}_2} \cup \tilde{\mathcal{R}}_{\mathrm{s}}$), d leaves from d stars (or $d-2$ leaves of $d-2$ stars) in ρ are mapped to the remaining d (or $d-2$) vertices of the path in $\tilde{\rho}$; the remaining leaves of the stars in ρ are mapped to the leaves of the corresponding stars in $\tilde{\rho}$.

Finally, for $j = 1, \ldots, n$, the paths $\pi_{j_1}, \pi_{j_2}, \pi_{j_3} \in \mathcal{P}$, where $1 \leq j_1, j_2, j_3 \leq 3n$, that correspond to the elements of $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$, are mapped to the $B/2$ length-1 paths from $\tilde{\mathcal{P}}_j$: The vertices with odd indices of path π_i, $i = \{j_1, j_2, j_3\}$ are mapped to the vertices of $\tilde{\mathcal{P}}_j$ that are adjacent to the leaves $\mathcal{L}_{\mathrm{m}_1}$ of $\tilde{S}_0$.

Next, we show that for the vertices of the paths in $\mathcal{P}$ the mapping f is compactness-preserving. Let $p^* \in V(\mathcal{P})$, z be the center vertex of the donor tree adjacent to p^*, and p' and $p'' \in V(\mathcal{P})$ be the other two neighbors of p^* (the similar will hold if p^* has only two neighbors). For the image and proper distances of p^* we have $L'_{p^*} = \mathrm{dist}_{T_2}(f(p^*), f(z)) + \mathrm{dist}_{T_2}(f(p^*), f(p')) + \mathrm{dist}_{T_2}(f(p^*), f(p''))$ and $L_{p^*} = 3$. Consider first p' is a leaf of S_0. If $f(p') \in \mathcal{L}_{\mathrm{m}_1}$, then $L'_{p^*} - L_{p^*} = ((d-1)+1+1) - 3 = d-2 < d$; otherwise, that is, $f(p') \in \mathcal{L}_{\mathrm{m}_2}$, we have $L'_{p^*} - L_{p^*} = ((d-1)+3+1) - 3 = d$. Consider now $p' \in V(\mathcal{P})$. If $f(p^*)$ is adjacent to a vertex in $\mathcal{L}_{\mathrm{m}_1}$, then $L'_{p^*} - L_{p^*} = ((d-1)+3+1) - 3 = d$; otherwise, $L'_{p^*} - L_{p^*} = ((d-1)+1+3) - 3 = d$.

"$\Longleftarrow$": Let f be a compactness-preserving mapping from T_1 to T_2. The next claims follow directly from the image distance constraints and the degrees of vertices (see Lemma 28) of the donor trees in D_{b}, $\mathcal{D}_{\mathrm{m}}$, $\mathcal{D}_{\mathrm{s}}$.
Claim 29.1: The vertices of D_{b} have to be mapped to the vertices of $\tilde{R}_{\mathrm{b}}$.

Proof. According to Lemma 28, the centers of the stars of D_{b} in T_1 have to be mapped to the centers of the stars in $\tilde{R}_{\mathrm{b}}$ in T_2 and the center vertex of D_{b} to the center vertex of $\tilde{R}_{\mathrm{b}}$. Then, there are d leaves in d stars from D_{b} that can only be mapped to the remaining vertices of the path in $\tilde{R}_{\mathrm{b}}$, since otherwise $L'_v - L_v > d$, where v is the center of a star in D_{b}. □

Since the distance between the center vertex of $\tilde{R}_{\mathrm{b}}$ and $\tilde{c}$ is $d+1$, the following is true:
Claim 29.2: The vertices of S_0 have to be mapped to the vertices of $\tilde{S}_0$ with $\tilde{c} = f(c)$.

Similarly, by Lemma 28 and Claims 29.1-29.2, we derive the following two claims:
Claim 29.3: The vertices of trees from $\mathcal{D}_{\mathrm{m}}$ have to be mapped to the vertices in $\tilde{\mathcal{R}}_{\mathrm{m}_1}$ and $\tilde{\mathcal{R}}_{\mathrm{m}_2}$.
Claim 29.4: The centers of the stars of a tree from $\mathcal{D}_{\mathrm{s}}$ have to be mapped to the centers of the stars of a tree from $\tilde{\mathcal{R}}_{\mathrm{s}}$.
Claim 29.5: Let $p_1 \in V(\mathcal{P})$ be adjacent to a leaf p_0 of S_0. If a compactness-preserving mapping f from T_1 to T_2 exists, then $f(p_1) \in V(\tilde{\mathcal{P}})$ and $f(p_1)$ is adjacent to vertices from $\mathcal{L}_{\mathrm{m}_1}$.

Proof. Consider the three neighbors of p_0: the center c of S_0, $z \in V(\mathcal{D}_{\mathrm{m}})$ and $p_1 \in V(\mathcal{P})$ (see Fig. 5.2). The image distance L'_{p_0} is equal to $\mathrm{dist}_{T_2}(f(p_0), f(c)) + \mathrm{dist}_{T_2}(f(p_0), f(z)) + \mathrm{dist}_{T_2}(f(p_0), f(p_1))$. If $f(p_0) \in \mathcal{L}_{\mathrm{m}_1}$, then $L'_{p_0} \geq 1 + (d +$

$1) + \mathrm{dist}_{T_2}(f(p_0), f(p_1))$ and $L'_{p_0} - L_{p_0} \geq 1 + (d+1) + \mathrm{dist}_{T_2}(f(p_0), f(p_1)) - 3 = \mathrm{dist}_{T_2}(f(p_0), f(p_1)) + d - 1$, which implies $\mathrm{dist}_{T_2}(f(p_0), f(p_1)) \leq 1$. Thus, $f(p_1)$ has to be adjacent to $f(p_0)$. Further, if $f(p_0) \in \mathcal{L}_{\mathrm{m}_2}$, then $L'_{p_0} = 1 + (d - 1) + \mathrm{dist}_{T_2}(f(p_0), f(p_1))$ and $L'_{p_0} - L_{p_0} \geq 1 + (d-1) + \mathrm{dist}_{T_2}(f(p_0), f(p_1)) - 3 = \mathrm{dist}_{T_2}(f(p_0), f(p_1)) + d - 3$, which implies $\mathrm{dist}_{T_2}(f(p_0), f(p_1)) \leq 3$. However, the vertices within distance three to $f(p_0)$ are either from $\tilde{S}_0$, $\tilde{R}_{\mathrm{b}}$ and the trees $\tilde{\mathcal{R}}_{\mathrm{m}_1} \cup \tilde{\mathcal{R}}_{\mathrm{m}_2}$ or the vertices in $\tilde{\mathcal{P}}$, which are adjacent to vertices from $\mathcal{L}_{\mathrm{m}_1}$. The claim follows then from Claims 29.1-29.3. □

Claim 29.6: Let $p_1 \in \mathcal{P}$ be adjacent to a leaf p_0 of S_0. If a compactness-preserving mapping f from T_1 to T_2 exists, then for every $v \in \mathcal{V}_1$, it holds $f(v) \in \mathcal{V}_2$, where $\mathcal{V}_1 := V(T_1(p_0, c)) \setminus \{p_0\}$ and $\mathcal{V}_2 := V(T_2(\tilde{p}_0, \tilde{c})) \setminus \{\tilde{p}_0\}$ with $\tilde{p}_0$ being a leaf of $\tilde{S}_0$ adjacent to $f(p_1)$.

Proof. We first show that the claim holds for every $p^* \in V(\mathcal{P}) \cap \mathcal{V}_1$. Let $z \in V(\mathcal{D}_{\mathrm{s}}), p', p''$ be the three neighbors of p^* (the similar will hold if p^* has only two neighbors). Suppose the claim is not true. Then, the image of at least one neighbor of p^* is not in $\mathcal{V}_2$.

First consider $f(z) \notin \mathcal{V}_2$. Since z is the center of the donor tree, which can only be mapped to the center of a recipient tree in $\tilde{\mathcal{R}}_{\mathrm{s}}$, the distance between $f(p^*)$ and $f(z)$ is at least $d+2$. Assuming that $f(p')$ and $f(p'')$ are adjacent to $f(p^*)$, we have $L'_{p^*} = \mathrm{dist}_{T_2}(f(p^*), f(z)) + \mathrm{dist}_{T_2}(f(p^*), f(p')) + \mathrm{dist}_{T_2}(f(p^*), f(p'')) \geq (d+2) + 1 + 1$, and $L'_{p^*} - L_{p^*} \geq (d+4) - 3 > d$, which is a contradiction.

Consider now that one of $f(p')$ and $f(p'')$ is not in $\mathcal{V}_2$, say $f(p') \notin \mathcal{V}_2$. Since z is the center of the donor tree, we have $\mathrm{dist}_{T_2}(f(p^*), f(z)) + \mathrm{dist}_{T_2}(f(p^*), f(p')) \geq \mathrm{dist}_{T_2}(f(z), f(p'))$, with the equality holding when $f(p^*)$ lies on the path between $f(z)$ and $f(p')$, and $\mathrm{dist}_{T_2}(f(z), f(p')) \geq d+3$. Then $L'_{p^*} - L_{p^*} \geq (d+3) + 1 - 3 > d$, which is a contradiction. Thus, the claim is true for every vertex p^* and the center z of the donor tree.

Let x be the center of a star adjacent to z. Suppose $f(x) \notin \mathcal{V}_2$. Since x can only be mapped to the center of a star in a recipient tree from $\tilde{\mathcal{R}}_{\mathrm{s}}$, the distance between $f(z)$ and $f(x)$ is at least $2d+3$. Assuming that the images of the remaining neighbors of x are adjacent to $f(x)$ in T_2, we have $L'_x - L_x \geq 2d+1 > d$, which is a contradiction.

Let y be a leaf adjacent to x. Suppose now $f(y) \notin \mathcal{V}_2$. Then, the distance between $f(y)$ and $f(x)$ takes the smallest value, when y is mapped to the vertex adjacent to $\mathcal{L}_{\mathrm{m}_1} \setminus \{\tilde{p}_0\}$, that is, $\mathrm{dist}_{T_2}(f(y), f(x)) \geq d+3$. Consequently, we have $L'_y - L_y \geq d+2 > d$, which is a contradiction. Thus, the claim is true. □

Note that $|\mathcal{V}_2| = 2 \cdot B/2 \cdot (1 + |\mathrm{RT}_1(Bdn^2, Bdn^3, d-1)|) = B \cdot (1 + |\mathrm{DT}_1(Bdn^2, Bdn^3)|)$. That is, $B \cdot |\mathrm{RT}_1(Bdn^2, Bdn^3, d-1)|$ vertices in $\mathcal{V}_2$ have to be mapped to B donor trees in $\mathcal{D}_{\mathrm{s}}$, each having $|\mathrm{DT}_1(Bdn^2, Bdn^3)|$ vertices. Consequently, there will remain B vertices in $\mathcal{V}_2$ that can only be mapped to the B vertices of

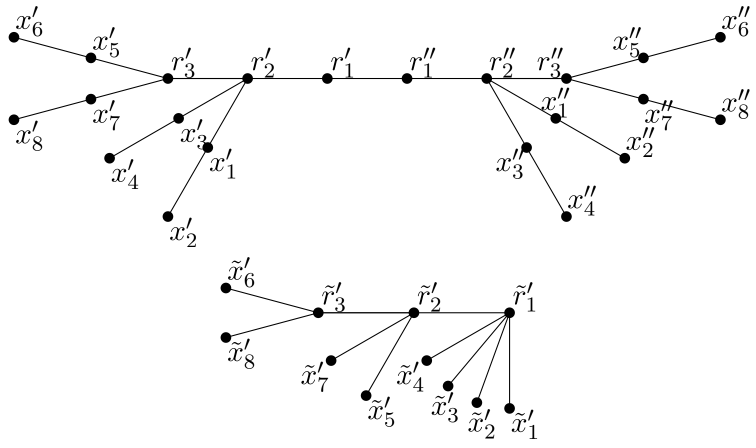


Fig. 5.3: A donor tree DT_2 and a recipient tree RT_2 defined in the proof of Thm. 30

three paths $\pi_{j_1}, \pi_{j_2}, \pi_{j_3} \in \mathcal{P}$, where $1 \le j_1, j_2, j_3 \le 3n$, with $|V(\pi_{j_1})| + |V(\pi_{j_2})| + |V(\pi_{j_3})| = B$. Thus, given the mapping f, we can derive the corresponding 3-partition for (A, B). □

5.1.2 Case $l = 2$, $d \ge 2$

Theorem 30. *CPM on trees with $l = 2$ and $d \ge 2$ is NP-hard.*

Proof. We consider only the case $d = 2$. The same can be shown for $d > 2$ by subdividing edges of the trees constructed in the following reduction.

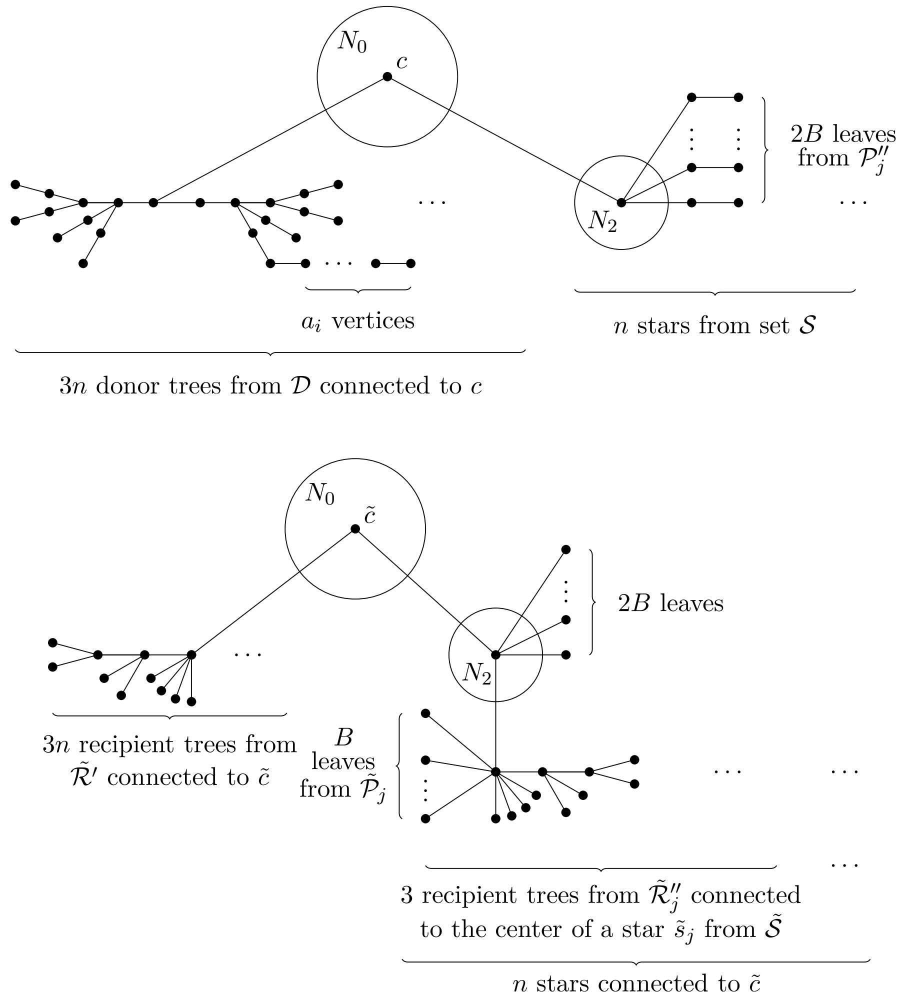
We reduce again from 3-Partition. We first introduce two gadgets used in the proof. A donor tree $T := \mathrm{DT}_2$ is a tree as depicted in Fig. 5.3. The subtrees $\rho' := T(r'_1, r''_1)$ and $\rho'' := T(r''_1, r'_1)$ are called the first and the second component subtrees, respectively. We call the vertex r'_1 the *root* of ρ'; the vertices r''_1 and x''_4 are called the *root* and the *tail* of ρ'', respectively. A recipient tree $\tilde{T} := \mathrm{RT}_2$ is a tree as depicted in Fig. 5.3. The vertex $\tilde{r}'_1$ is called the root of $\tilde{T}$.

The tree T_1 is composed of the following components:

- a star S_0 with N_0 leaves and center c,
- a set $\mathcal{S}$ containing n stars, each having $N_1 + 2B$ leaves,
- a set $\mathcal{D}$ containing $3n$ donor trees DT_2,
- a set $\mathcal{P}'$ with $3n$ paths; the i-th path corresponds to one particular element a_i of A and has a_i vertices, $i = 1, \ldots, 3n$,
- a set $\mathcal{P}'' := \bigcup_{j=1}^{n} \mathcal{P}''_j$, where $\mathcal{P}''_j$ is a set of $2B$ vertices,

where $N_0 := Bn^3, N_1 := Bn$.

The components are connected to T_1 as follows (see Fig. 5.4). The root of the first component subtree of every donor tree $\rho \in \mathcal{D}$ is made adjacent to c;



N_0
c
$2B$ leaves from $\mathcal{P}''_j$
N_2
a_i vertices
n stars from set $\mathcal{S}$
$3n$ donor trees from $\mathcal{D}$ connected to c
N_0
$\tilde{c}$
$2B$ leaves
N_2
$3n$ recipient trees from $\tilde{\mathcal{R}}'$ connected to $\tilde{c}$
B leaves from $\tilde{\mathcal{P}}_j$
3 recipient trees from $\tilde{\mathcal{R}}''_j$ connected to the center of a star $\tilde{s}_j$ from $\tilde{\mathcal{S}}$
n stars connected to $\tilde{c}$

Fig. 5.4: The trees T_1 and T_2 of CPM on trees corresponding to an instance of 3-Partition as defined in the proof of Thm. 30

the second component subtree of ρ is connected to one distinct path from $\mathcal{P}'$ by adding an edge between the tail and an end-vertex of the path. For every star $s_j \in \mathcal{S}$, $j = 1, \dots, n$, the center of s_j is made adjacent to c; every vertex from $\mathcal{P}''_j$ is made adjacent to one distinct leaf in s_j.

The tree T_2 is composed of the following components:

- a star $\tilde{S}_0$ with N_0 leaves and center $\tilde{c}$,
- a set $\tilde{\mathcal{S}}$ containing n stars, each having $N_1 + 2B$ leaves,
- a set $\tilde{\mathcal{R}}'$ containing $3n$ recipient trees RT_2,
- a set $\tilde{\mathcal{R}}'' := \bigcup_{j=1}^{n} \tilde{\mathcal{R}}''_j$, where $\tilde{\mathcal{R}}''_j$ contains 3 recipient trees RT_2, and

Tab. 5.2: Detailed computations for a mapping from a component subtree of a donor tree DT$_2$ to a recipient tree RT$_2$ in the proof of Thm. 30

v	$\lvert N^2_{T_1}(v)\rvert$	L_v	L'_v	detailed computation of L'_v
r'_1	$7+\mathcal{N}$	$3+2\cdot(4+\lvert\mathcal{N}\rvert) = 11+2\lvert\mathcal{N}\rvert$	$13+2\mathcal{N}$	$x'_1\ x'_3\ \ r'_2\ \ r'_3\ \ c\ \ \lvert\mathcal{N}\rvert\ \ r''_1\ \ r''_2$ $1+1+1+2+1+2\lvert\mathcal{N}\rvert+3+4$
r'_2	10	$4+2\cdot 6 = 16$	18	$r'_1\ \ r'_3\ \ x'_1\ \ x'_3\ \ x'_2\ x'_4\ \ x'_5\ \ x'_7\ \ c\ \ r''_1$ $1+1+2+2+2+2+1+1+2+4$
r'_3	8	$3+2\cdot 5 = 13$	15	$r'_2\ \ r'_1\ \ x'_5\ \ x'_7\ \ x'_6\ \ x'_8\ \ x'_1\ \ x'_3$ $1+2+2+2+1+1+3+3$
$x'_1(x'_3)$	5	$2+2\cdot 3 = 8$	10	$x'_2\ \ r'_2\ \ r'_1\ \ r'_3\ \ x'_3$ $2+2+1+3+2$
$x'_2(x'_4)$	2	$1+2\cdot 1 = 3$	4	$x'_1\ \ r'_2$ $2+2$
$x'_5(x'_7)$	4	$2+2\cdot 2 = 6$	8	$r'_3\ \ x'_6\ \ r'_2\ \ x'_7$ $2+3+1+2$
$x'_6(x'_8)$	2	$1+2\cdot 1 = 3$	4	$x'_5\ \ r'_3$ $3+1$

- a set $\tilde{\mathcal{P}} := \bigcup_{j=1}^{n} \tilde{\mathcal{P}}_j$, where $\tilde{\mathcal{P}}_j$ is a set of $3B$ vertices.

The components are combined to T_2 as follows (see Fig. 5.4). The roots of the recipient trees from $\tilde{\mathcal{R}}'$ are made adjacent to $\tilde{c}$. The center of each star $\tilde{s}_j \in \tilde{\mathcal{S}}$ for $i = 1, \dots, n$, is made adjacent to $\tilde{c}$ and the roots of three recipient trees from $\tilde{\mathcal{R}}''_j$. For every recipient tree in $\tilde{\mathcal{R}}''_j$, we add B edges between its root and B distinct vertices from $\tilde{\mathcal{P}}_j$.

"$\Longrightarrow$": Let $A_1, \dots, A_n$ be a 3-partition for an instance (A, B) of 3-Partition. The mapping f is constructed as follows. The vertices of S_0 are mapped to the vertices of $\tilde{S}_0$ with $\tilde{c} = f(c)$; the stars in $\mathcal{S}$ are mapped to the stars in $\tilde{\mathcal{S}}$ with leaves being mapped to leaves. The vertices in the first component subtrees of the donor trees are mapped to the vertices of the recipient trees in $\tilde{\mathcal{R}}'$ with $\tilde{r}'_j = f(r'_j)$, $j = 1, \dots, 3$ and $\tilde{x}'_i = f(x'_i)$, $i = 1, \dots, 8$ (see Fig. 5.3). Similarly, the vertices in the second component subtrees of the donor trees are mapped to the vertices in recipient trees in $\tilde{\mathcal{R}}''$.

For $j = 1, \dots, n$ and a set $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$ of the 3-partition, where $1 \leq j_1, j_2, j_3 \leq 3n$, the vertices of the corresponding paths $\pi_{j_1}, \pi_{j_2}, \pi_{j_2} \in \mathcal{P}'$ are mapped to the vertices from $\tilde{\mathcal{P}}_j$. Note that the vertices in $\tilde{\mathcal{P}}_j$ are adjacent to the roots of the three recipient trees in $\tilde{\mathcal{R}}''_j$. The $2B$ vertices in $\mathcal{P}'_j$ are mapped to the remaining vertices in $\tilde{\mathcal{P}}_j$.

Given the mapping f, the differences between the proper and image distances of the vertices in the first component subtrees of the donor trees in $\mathcal{D}$ are as follows: $L'_v - L_v = 2$, for $v \in \{r'_1, r'_2, r'_3, x'_1, x'_3, x'_5, x'_7\}$, and $L'_u - L_u = 1$, for $u \in \{x'_2, x'_4, x'_6, x'_8\}$. The same holds for the vertices of the second component subtrees and the vertices of the paths from $\mathcal{P}'$, which are connected to the second component subtrees. For the root r'_1 we have $N_{T_1}(r'_1) = \{c, r'_2, r''_1\}$. The exact 2-neighborhood of r'_1 consists of x'_1, x'_3, r'_3, r''_2 and the set $\mathcal{N}$ of $N_0 + 4n$ vertices adjacent to c. The differences between distances in T_2 and T_1 from r'_1 to its 2-neighbors are as follows (see Table 5.2 for the detailed computations): $\text{dist}_{T_2}(f(r'_1), f(c)) - \text{dist}_{T_1}(r'_1, c) = 1 - 1 = 0$, $\text{dist}_{T_2}(f(r'_1), f(r'_2)) - \text{dist}_{T_1}(r'_1, r'_2) = 1 - 1 = 0$, $\text{dist}_{T_2}(f(r'_1), f(r''_1)) - \text{dist}_{T_1}(r'_1, r''_1) = 3 - 1 = 2$, $\text{dist}_{T_2}(f(r'_1), f(x'_1)) - \text{dist}_{T_1}(r'_1, x'_1) = 1 - 2 = -1$, $\text{dist}_{T_2}(f(r'_1), f(x'_2)) - \text{dist}_{T_1}(r'_1, x'_2) = 1 - 2 = -1$, $\text{dist}_{T_2}(f(r'_1), f(r'_3)) - \text{dist}_{T_1}(r'_1, r'_3) = 2 - 2 = 0$, $\text{dist}_{T_2}(f(r'_1), f(r''_2)) - \text{dist}_{T_1}(r'_1, r''_2) = 4 - 2 = 2$, and, for every $v \in \mathcal{N}$, $\text{dist}_{T_2}(f(r'_1), f(v)) - \text{dist}_{T_1}(r'_1, v) = 2 - 2 = 0$. Thus, we have $L'_{r'_1} - L_{r'_1} = 0 + 0 + 2 - 1 - 1 + 0 + 2 + |\mathcal{N}| \cdot 0 = 2 = d$. Consider now a path $\pi = \{p_1, \ldots, p_{|V(\pi)|}\} \in \mathcal{P}$ and the tail x''_4 of the donor tree connected to π. All 2-neighbors $\{x''_3, p_1, r''_2, p_2\}$ of x''_4 are mapped to vertices with distance two from $f(x''_4)$. Thus, $L'_{x''_4} - L_{x''_4} = 2 \cdot |N^2_{T_1}(x''_4)| - (|N_{T_1}(x''_4)| + 2 \cdot |\hat{N}^2_{T_1}(x''_4)|) = 2 = d$. A similar argument applies to all vertices of π.

"$\Longleftarrow$": Let f be a compactness-preserving mapping from T_1 to T_2. The next claims follow from the sizes of 2-neighborhoods of the vertices.

Claim 30.1: The vertices of S_0 have to be mapped to $\tilde{S}_0$ with $\tilde{c} = f(c)$.

Claim 30.2: The vertices of a star in $\mathcal{S}$ have to be mapped to the vertices of a star in $\tilde{\mathcal{S}}$ with centers being mapped to centers.

Claim 30.3: The root of the first component subtree of a donor tree in $\mathcal{D}$ has to be mapped to the root of the recipient tree in $\tilde{\mathcal{R}}'$.

Now we prove the following:

Claim 30.4: The first component subtree ρ' of a donor tree $\rho \in \mathcal{D}$ has to be mapped to a recipient tree $\tilde{\rho}' \in \tilde{\mathcal{R}}'$.

Proof. In the proof we first show that, if a leaf vertex v in ρ' is mapped to $\tilde{\rho}'$, then v's 2-neighborhood has to be mapped to $\tilde{\rho}'$ (see Fig. 5.3 and Fig. 5.4). Then, we prove that the vertices r'_2 and r'_3 also have to be mapped to ρ'. Finally, we show that the images of the vertex r'_1 also cannot be far from the images of its 2-neighbors and have to be mapped to $\tilde{\rho}'$.

Let r'_j, r''_j, $j = 1, \ldots, 3$, and x'_i, x''_i, $i = 1, \ldots, 8$ be the vertices in ρ' and the second component subtree ρ'' of ρ as depicted in Fig. 5.3.

Subclaim 30.5: If $f(x'_2) \in V(\tilde{\rho}')$, then $f(x'_1) \in V(\tilde{\rho}')$ and $f(r'_2) \in V(\tilde{\rho}')$.

Proof. Suppose the subclaim is not true, say $f(x'_1) \notin V(\tilde{\rho}')$. Then according to Claims 30.1-30.3, $\text{dist}_{T_2}(f(x'_2), f(x'_1)) \geq 4$. Thus, $L'_{x'_2} - L_{x'_2} \leq 2$ implies that $f(r'_2)$ has to be adjacent to $f(x'_2)$. Then, assuming that the images of the remaining 2-neighbors of x'_1 are adjacent to $f(x'_1)$ in T_2, we have $L'_{x'_1} \geq \text{dist}_{T_2}(f(x'_1), f(x'_2)) + \text{dist}_{T_2}(f(x'_1), f(r'_2)) + (|N^2_{T_1}(x'_1)| - 2) \geq 4 + 5 + (5 - 2) = 12$. Consequently, $L'_{x'_1} - L_{x'_1} \geq 4 > 2 = d$, which is a contradiction.

Assume now $f(r'_2) \notin V(\tilde{\rho}')$. Then, $\mathrm{dist}_{T_2}(f(x'_2), f(r'_2)) \geq 4$. Thus, $L'_{x'_2} - L_{x'_2} \leq 2$ implies that $f(x'_1)$ has to be adjacent to $f(x'_2)$. Then $L'_{x'_1} = \mathrm{dist}_{T_2}(f(x'_1), f(x'_2)) + \mathrm{dist}_{T_2}(f(x'_1), f(r'_2)) + \mathrm{dist}_{T_2}(f(x'_1), f(r'_1)) + \mathrm{dist}_{T_2}(f(x'_1), f(r'_3)) + \mathrm{dist}_{T_2}(f(x'_1), f(x'_3)) \geq 1+5+2+1+1 = 10$. Consequently, $L'_{x'_1} - L_{x'_1} \geq 2 = d$ implies that the vertices $f(r'_3)$, $f(x'_3)$ have to be adjacent to $f(x'_1)$ and $f(r'_1) \in V(\tilde{\rho}')$. Then, assuming that the images of the remaining 2-neighbors of r'_2 are adjacent to $f(r'_2)$ in T_2, we have $L'_{r'_2} \geq \mathrm{dist}_{T_2}(f(r'_2), f(r'_1)) + \mathrm{dist}_{T_2}(f(r'_2), f(x'_1)) + \mathrm{dist}_{T_2}(f(r'_2), f(x'_2)) + \mathrm{dist}_{T_2}(f(r'_2), f(x'_3)) + \mathrm{dist}_{T_2}(f(r'_2), f(r'_3)) + \mathrm{dist}_{T_2}(f(r'_2), f(c)) + (|N^2_{T_1}(r'_2)| - 6) = 3+5+4+6+6+2+(10-6) = 30$ and $L'_{r'_2} - L_{r'_2} \geq 15$, a contradiction. Thus, the subclaim is true. □

Obviously, similar claims hold for x'_4, x'_6 and x'_8.

Subclaim 30.6: If $f(r'_3) \in V(\tilde{\rho}')$, then $f(r'_2) \in V(\tilde{\rho}')$.

Proof. Suppose the subclaim is not true. Then, assuming that the images of the remaining 2-neighbors of r'_3 are adjacent to $f(r'_3)$ in T_2, we have $L'_{r'_3} \geq \mathrm{dist}_{T_2}(f(r'_3), f(r'_2)) + \mathrm{dist}_{T_2}(f(r'_3), f(x'_1)) + \mathrm{dist}_{T_2}(f(r'_3), f(x'_3)) + (|N^2_{T_1}(r'_3)| - 3) \geq 4+4+4+(8-3) = 17$ and $L'_{r'_3} - L_{r'_3} \geq 4 > 2 = d$, which is a contradiction. □

Subclaim 30.7: If $f(r'_2) \in V(\tilde{\rho}')$, then $f(r'_1) \in V(\tilde{\rho}')$.

Proof. Suppose the subclaim is not true, i.e. $f(r'_1) \notin V(\tilde{\rho}')$. Then, assuming that the images of 2-neighbors of r'_1 in $N^2_{T_1}(r'_1) \setminus N_{T_1}(c)$ are adjacent to $f(r'_1)$ in T_2, we have $L'_{r'_1} \geq \mathrm{dist}_{T_2}(f(r'_1), f(r'_2)) + \mathrm{dist}_{T_2}(f(r'_1), f(r'_3)) + \mathrm{dist}_{T_2}(f(r'_1), f(x'_1)) + \mathrm{dist}_{T_2}(f(r'_1), f(x'_3)) + 2 \cdot |N_{T_1}(c)| + (N^2_{T_1}(r'_1) \setminus N_{T_1}(c) - 4) \geq 3+3+3+3+2 \cdot (N_0 + 4n) + (7-4) = 15 + 2 \cdot (N_0 + 4n)$ and $L'_{r'_1} - L_{r'_1} \geq 4 > 2 = d$, which is a contradiction. □

Summarizing the above three subclaims, we can conclude that ρ' has to be mapped to $\tilde{\rho}'$. □

Now observe that, by Claim 30.4 and $|\mathrm{DT}_2| = 2 \cdot |\mathrm{RT}_2|$, no vertices of the second component subtrees of the donor trees can be mapped to the recipient trees in $\tilde{\mathcal{R}}'$.

The above subclaims can be applied to the second component subtrees of the donor trees in $\mathcal{D}$ and the recipient trees in $\tilde{\mathcal{R}}''$. That is, the following two claims are true.

Claim 30.8: The second component subtree of a donor tree from $\mathcal{D}$ has to be mapped to a recipient tree in $\tilde{\mathcal{R}}''$.

Claim 30.9: Let ρ'' be the second component subtree of a donor tree from $\mathcal{D}$ with ρ'' being mapped to $\tilde{\rho}'' \in \tilde{\mathcal{R}}''$. The path from $\mathcal{P}'$ connected to ρ'' has to be mapped to the vertices from $\tilde{\mathcal{P}}$ that are adjacent to the root of $\tilde{\rho}''$.

It remains to consider the vertices in $\mathcal{P}''$.

Claim 30.10: Let $s_j \in \mathcal{S}$, $j = 1, \ldots, n$, and c_j be the center of s_j with $\tilde{c}_j = f(c_j)$, where $\tilde{c}_j$ is the center of $\tilde{s}_j \in \tilde{S}$. The $2B$ leaves in $\mathcal{P}''_j$ have to be mapped to $\tilde{\mathcal{P}}_j$.

Let the paths $\pi_{j_1}, \pi_{j_2}, \pi_{j_3} \in \mathcal{P}'$ that correspond to the elements $a_{j_1}, a_{j_2}, a_{j_3} \in A$, where $1 \leq j_1, j_2, j_3 \leq 3n$, be mapped to the vertices from $\tilde{\mathcal{P}}_j$, $1 \leq j \leq n$.

Suppose $a_{j_1} + a_{j_2} + a_{j_3} > B$. Then, there exists at least one leaf in $\mathcal{P}''_j$ that cannot be mapped to $\tilde{\mathcal{P}}_j$, a contradiction to Claim 30.10.

Suppose now $a_{j_1} + a_{j_2} + a_{j_3} < B$. Then, there exist $a_{j_4}, a_{j_5}, a_{j_6} \in A$, where $1 \leq j_4, j_5, j_6 \leq 3n$, with $a_{j_4} + a_{j_5} + a_{j_6} > B$ that are mapped to $\tilde{\mathcal{P}}_{j'}$, $1 \leq j' \leq n$, which is a contradiction to Claim 30.10.

Thus, the mapping f exists only if for every $\pi_{j_1}, \pi_{j_2}, \pi_{j_3} \in \mathcal{P}'$ that are mapped to $\tilde{\mathcal{P}}_j$ and correspond to $a_{j_1}, a_{j_2}, a_{j_3} \in A$, we have $a_{j_1} + a_{j_2} + a_{j_3} = B$. Consequently, given the mapping f, we can derive the corresponding 3-partition for the instance (A, B). □

5.1.3 Case $l \geq 3$, $d \geq 0$

Theorem 31. *CPM on trees with $l = 3$ and $d \geq 0$ is NP-hard.*

Proof. We reduce again from 3-Partition. We first introduce three graph gadgets, and indicate their relation to CPM on trees. The key for the reduction is the observation that the diameter of a star is at most 2. For a vertex $v \in T_1$ with a large exact 3-neighborhood $\hat{N}^3_{T_1}(v)$, we can map v together with some vertices in $\hat{N}^3_{T_1}(v)$ to a star. By doing this, we will have L'_v much smaller than L_v with respect to v and these vertices from $\hat{N}^3_{T_1}(v)$, allowing us to map other vertices from the 3-neighborhood of v to vertices in T_2 far from v's image.

Consider integers $k_1, \ldots, k_5$ and Δ, where $k_1 := 1$, $k_2 = 2$, $k_4 := 2$, $k_5 := 5$, Δ is much greater than $\max\{k_1, k_2, k_4, k_5\}$, and $k_3 := \Delta - 2$. We further define $K(\Delta) := 1 + k_1 \cdot (1 + k_2 \cdot (1 + k_3 \cdot (1 + k_4 \cdot (1 + k_5))))$.

A *component subtree* T' consists of $K(\Delta)$ vertices (see Fig. 5.5). The root of T' has $k_1 = 1$ child also called "level-2" vertex. Every "level-i" vertex has as children k_i-many "level-$(i+1)$" vertices, $i = 2, \ldots, 5$. We denote the set of level-i vertices in T' by $\mathcal{L}_i(T')$. A *donor tree* $\mathrm{DT}_3(\Delta)$ consists of two copies of the component subtree: The two component subtrees are connected by adding an edge between their roots. A *recipient star* $\mathrm{RS}_3(\Delta)$ is a star with $K(\Delta)$ vertices including the center. A *donor path* $P := \mathrm{DP}_3(a, b)$ consists of a star with a leaves and a path of length b. One end-vertex of the path is made adjacent to the center of the star. There are two additional "special" vertices in P that are connected to two arbitrary leaves of the star. A *recipient path* $\mathrm{RP}_3(a, b)$ consists of a star with $a + 2$ leaves and a path of length b. One end-vertex of the path is made adjacent to the center of the star. In a donor path or a recipient path, the *tail* is the end-vertex of the path that is not adjacent to the star.

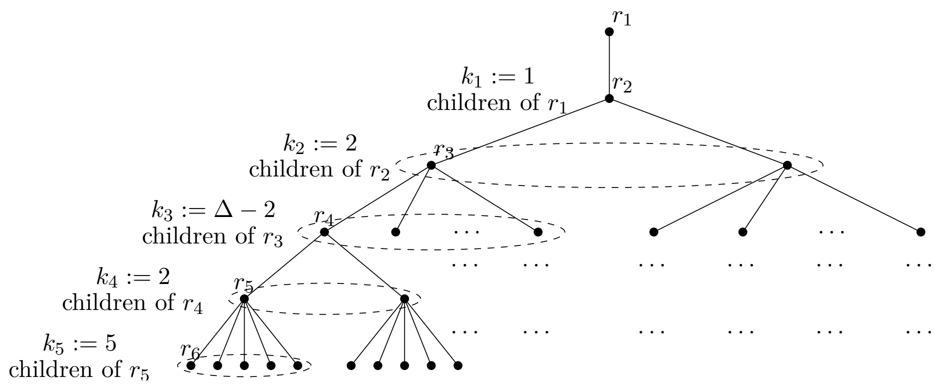


Fig. 5.5: A component subtree of a donor tree $\mathrm{DT}_3(\Delta)$ defined in the proof of Thm. 31

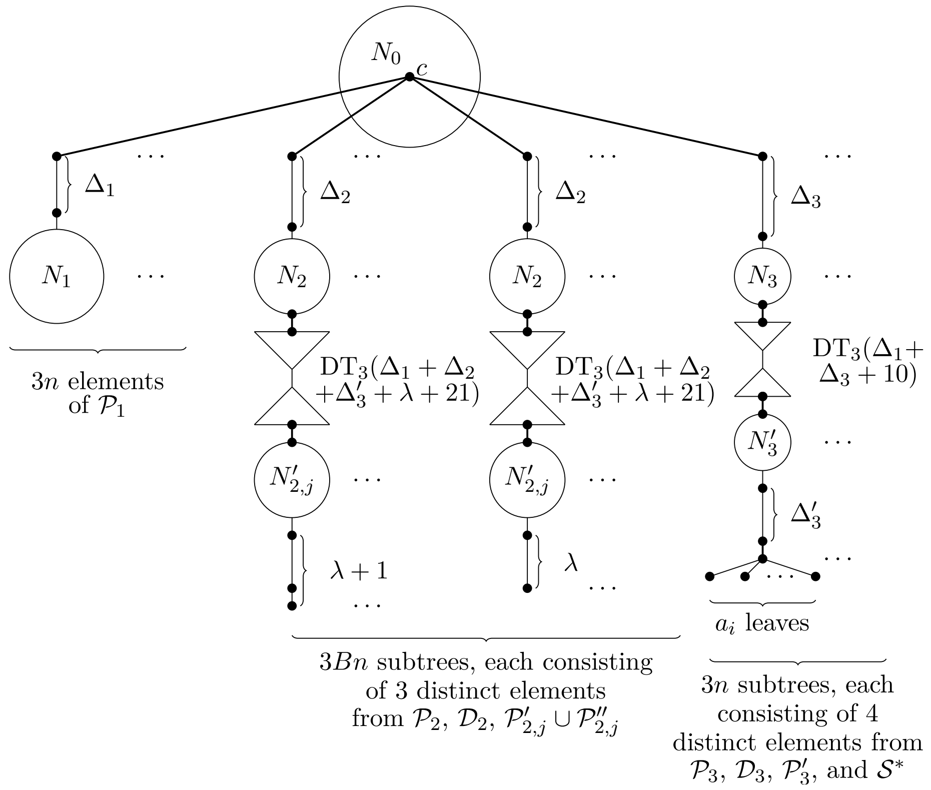
Now we construct trees T_1 and T_2 corresponding to an instance (A, B) of 3-Partition.

The tree T_1 consists of the following components (see Fig. 5.6):

- one "big" star S_0 with N_0 leaves and center c,
- a set $\mathcal{P}_1$ of $3n$ donor paths $\mathrm{DP}_3(N_1, \Delta_1)$,
- a set $\mathcal{P}_2$ of $3Bn$ donor paths $\mathrm{DP}_3(N_2, \Delta_2)$,
- for each $j = 1, \ldots, n$, a set $\mathcal{P}'_{2,j}$ of $2B$ donor paths $\mathrm{DP}_3(N'_{2,j}, \lambda + 1)$,
- for each $j = 1, \ldots, n$, a set $\mathcal{P}''_{2,j}$ of B donor paths $\mathrm{DP}_3(N'_{2,j}, \lambda)$,
- a set $\mathcal{P}_3$ of $3n$ donor paths $\mathrm{DP}_3(N_3, \Delta_3)$,
- a set $\mathcal{P}'_3$ of $3n$ donor paths $\mathrm{DP}_3(N'_3, \Delta'_3)$,
- a set $\mathcal{D}_2$ of $3Bn$ donor trees $\mathrm{DT}_3(\Delta_1 + \Delta_2 + \Delta'_3 + \lambda + 21)$,
- a set $\mathcal{D}_3$ of $3n$ donor trees $\mathrm{DT}_3(\Delta_1 + \Delta_3 + 10)$, and
- a set $\mathcal{S}^*$ of $3n$ stars; the i-th star for $i = 1, \ldots, 3n$ corresponds to the element a_i of A and has a_i leaves.

Here $N_0 := Bn^{10}$, $N_1 := Bn^9$, $N_2 := Bn^8$, $N'_{2,j} := jBn^6$, for $j := 1, \ldots, n$, $N_3 := Bn^5$, $N'_3 := Bn^4$, $\Delta_1 := Bn^3$, $\Delta_2 := Bn^2$, $\Delta_3 := Bn$, $\Delta'_3 := 2Bn$, and $\lambda := 3Bn$.

The components are combined to T_1 as follows (see Fig. 5.6). The tails of donor paths in $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ are made adjacent to c. Then, we build three one-to-one matchings: $\mathcal{G}_1$ between $\mathcal{P}_2$, $\mathcal{D}_2$ and $\bigcup_{j=1}^{n}(\mathcal{P}'_{2,j} \cup \mathcal{P}''_{2,j})$, $\mathcal{G}_2$ between $\mathcal{P}_3$ and $\mathcal{D}_3$, $\mathcal{G}_3$ between $\mathcal{P}'_3$ and $\mathcal{S}^*$. For $\mathcal{G}_1$, we connect a donor tree $\rho \in \mathcal{D}_2$ to a donor path $\pi \in \mathcal{P}_2$ and a donor path $\pi' \in \bigcup_{j=1}^{n}(\mathcal{P}'_{2,j} \cup \mathcal{P}''_{2,j})$. We add an edge between one arbitrary level-6 vertex of the first component subtree of ρ and a leaf vertex of the star of π. Then, one arbitrary level-6 vertex of the second component subtree of ρ is made adjacent to a leaf vertex of the star of π'. For $\mathcal{G}_2$, a donor tree $\rho \in \mathcal{D}_3$ is connected to a donor path $\pi \in \mathcal{P}_3$ by adding an edge between one arbitrary level-6 vertex of ρ's first component subtree and a leaf vertex of the star of π. For $\mathcal{G}_3$, we connect a donor path $\pi' \in \mathcal{P}'_3$ to a star $s^* \in \mathcal{S}^*$ by adding



N_0
c
Δ_1
Δ_2
Δ_2
Δ_3
N_1
N_2
N_2
N_3
$3n$ elements of $\mathcal{P}_1$
$\mathrm{DT}_3(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$
$\mathrm{DT}_3(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$
$\mathrm{DT}_3(\Delta_1 + \Delta_3 + 10)$
$N_{2,j}'$
$N_{2,j}'$
N_3'
Δ_3'
$\lambda + 1$
λ
a_i leaves
$3Bn$ subtrees, each consisting of 3 distinct elements from $\mathcal{P}_2$, $\mathcal{D}_2$, $\mathcal{P}_{2,j}' \cup \mathcal{P}_{2,j}''$
$3n$ subtrees, each consisting of 4 distinct elements from $\mathcal{P}_3$, $\mathcal{D}_3$, $\mathcal{P}_3'$, and $\mathcal{S}^*$

Fig. 5.6: A schematic representation of T_1 of CPM on trees with $l = 3$ and $d = 0$ corresponding to an instance of 3-PARTITION. Circles and triangles correspond to the stars and component subtrees, respectively. Thick lines correspond to the edges that connect T_1's components.

an edge between the tail of π' and the center of s^*. We finalize the construction of T_1 by adding an edge between one level-6 vertex of the second component subtree of $\rho \in \mathcal{G}_2$ and a leaf vertex of the star of one particular donor path from $\mathcal{G}_3$. The two level-6 vertices of a donor tree $\rho \in \mathcal{D}_2 \cup \mathcal{D}_3$, connecting two components in T_1, are called *tails* of ρ.

The tree T_2 consists of the following components (see Fig. 5.7):

- one "big" star $\tilde{S}_0$ with N_0 leaves and center $\tilde{c}$,
- a set $\tilde{\mathcal{P}}_1$ of $3n$ recipient paths $\mathrm{RP}_3(N_1, \Delta_1)$,
- a set $\tilde{\mathcal{P}}_2$ of $3Bn$ recipient paths $\mathrm{RP}_3(N_2, \Delta_2)$,
- for each $j = 1, \ldots, n$, a set $\tilde{\mathcal{P}}_{2,j}'''$ of $3B$ recipient path $\mathrm{RP}_3(N_{2,j}', \lambda)$,
- a set $\tilde{\mathcal{P}}_3$ of $3n$ recipient paths $\mathrm{RP}_3(N_3, \Delta_3)$,
- a set $\tilde{\mathcal{P}}_3'$ of $3n$ recipient paths $\mathrm{RP}_3(N_3', \Delta_3')$,
- a set $\tilde{\mathcal{R}}_2'$ of $3Bn$ recipient stars $\mathrm{RS}_3(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$,
- a set $\tilde{\mathcal{R}}_2''$ of $3Bn$ recipient stars $\mathrm{RS}_3(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$,

- a set $\tilde{\mathcal{R}}'_3$ of $3n$ recipient stars $\mathrm{RS}_3(\Delta_1 + \Delta_3 + 10)$,
- a set $\tilde{\mathcal{R}}''_3$ of $3n$ recipient stars $\mathrm{RS}_3(\Delta_1 + \Delta_3 + 10)$, and
- a set $\tilde{\mathcal{S}}^* = \bigcup_{j=1}^{n} \tilde{\mathcal{S}}^*_j$, where $\tilde{\mathcal{S}}^*_j$ is a set of three stars, each having B leaves.

The components are combined to T_2 as follows (see Fig. 5.7). The tails of the donor paths in $\tilde{\mathcal{P}}_1 \cup \tilde{\mathcal{P}}_2 \cup \tilde{\mathcal{P}}_3$ are made adjacent to $\tilde{c}$. We connect every recipient path $\tilde{\pi} \in \tilde{\mathcal{P}}_2$ to one distinct recipient star $\tilde{\rho} \in \tilde{\mathcal{R}}'_2$ by adding an edge between a leaf vertex of the star in $\tilde{\pi}$ and a leaf of $\tilde{\rho}$. The similar connections are made between $\bigcup_{j=1}^{n} \tilde{\mathcal{P}}'''_{2,j}$ and $\tilde{\mathcal{R}}''_2$, $\tilde{\mathcal{P}}_3$ and $\tilde{\mathcal{R}}'_3$, $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{R}}''_3$. Every recipient path $\tilde{\pi} \in \tilde{\mathcal{P}}'_3$ is connected to a recipient star $\tilde{\rho} \in \tilde{\mathcal{R}}'_3$ and a star $\tilde{s}^* \in \tilde{\mathcal{S}}^*$. We add one edge between a leaf vertex of $\tilde{\pi}$'s star and a leaf of $\tilde{\rho}$ and one edge between $\tilde{\pi}$'s tail and the center of $\tilde{s}^*$. Finally, for $j = 1, \dots, n$, the leaves of the three stars in $\tilde{\mathcal{S}}^*_j$ are made adjacent to the tails of the recipient paths in the set $\tilde{\mathcal{P}}'''_{2,j}$. In the following, for a recipient star $\tilde{\rho} \in \tilde{\mathcal{R}}'_2 \cup \tilde{\mathcal{R}}''_2 \cup \tilde{\mathcal{R}}'_3 \cup \tilde{\mathcal{R}}''_3$, we refer to the leaves of $\tilde{\rho}$ that are made adjacent to the other components of T_2 as the *tails* of $\tilde{\rho}$. Note that for a star from $\tilde{\mathcal{R}}'_2$ and a star from $\tilde{\mathcal{R}}''_2$ the distance between their centers is $\Delta_1 + \Delta_2 + \Delta'_3 + \lambda + 21$, while for a star from $\tilde{\mathcal{R}}'_3$ and a star from $\tilde{\mathcal{R}}''_3$ the distance between their centers is $\Delta_1 + \Delta_3 + 10$.

The time needed to construct T_1 and T_2 is clearly polynomial in n and B. Next we show that an instance of 3-Partition is a yes-instance iff the corresponding instance of CPM with T_1 and T_2 is a yes-instance.

"$\Longrightarrow$": Let $A_1, \dots, A_n$ be a 3-partition for an instance (A, B) of 3-Partition. The mapping f is constructed as follows. The star S_0 is mapped to $\tilde{S}_0$ with $\tilde{c} = f(c)$. Then a donor path $\pi \in \mathcal{P}_1$ is mapped to a recipient path $\tilde{\pi} \in \tilde{\mathcal{P}}_1$. For the path P of π, the i-th vertex, starting from π's tail, is mapped to the i-th vertex of the path in $\tilde{\pi}$, for $i = 1, \dots, |V(P)|$. The center of the star s in π is mapped to the center of the star $\tilde{s}$ in $\tilde{\pi}$; the leaves of s are mapped to the leaves of $\tilde{s}$. Note that two leaves of $\tilde{s}$ remain without mapped vertices: These two leaves are mapped to two special vertices in π. The similar mapping is constructed for donor paths in $\mathcal{P}_2$; they are mapped to recipient path $\tilde{\mathcal{P}}_2$; $\mathcal{P}_3$ to $\tilde{\mathcal{P}}_3$; $\mathcal{P}'_3$ to $\tilde{\mathcal{P}}'_3$.

The donor trees in $\mathcal{D}_2$ are mapped to the recipient stars in $\tilde{\mathcal{R}}'_2$ and $\tilde{\mathcal{R}}''_2$, while the donor trees in $\mathcal{D}_3$ are mapped to the recipient stars in $\tilde{\mathcal{R}}'_3$ and $\tilde{\mathcal{R}}''_3$. Here, for a donor tree ρ in $\mathcal{D}_2$ (or $\mathcal{D}_3$), the first component subtree of ρ is mapped to the corresponding recipient star in $\tilde{\mathcal{R}}'_2$ (or $\tilde{\mathcal{R}}'_3$); the second component subtree of ρ is mapped to a star in $\tilde{\mathcal{R}}''_2$ (or $\tilde{\mathcal{R}}''_3$); ρ's tails are mapped to the tails of the two corresponding recipient stars, the roots of ρ's component subtrees are mapped to centers of the stars.

Now consider, for $j = 1, \dots, n$, subset $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$ of the 3-partition, where $1 \leq j_1, j_2, j_3 \leq 3n$. The centers of the corresponding stars $s_{j_1}, s_{j_2}, s_{j_3} \in \mathcal{S}^*$ are mapped to the centers of the stars $\tilde{s}_{j_1}, \tilde{s}_{j_2}, \tilde{s}_{j_3} \in \tilde{\mathcal{S}}^*_j$. The leaves of s_{j_i} are mapped to the leaves of $\tilde{s}_{j_i}$, $i = 1, \dots, 3$. We denote these leaves by $\tilde{\Lambda}'_j$ and the

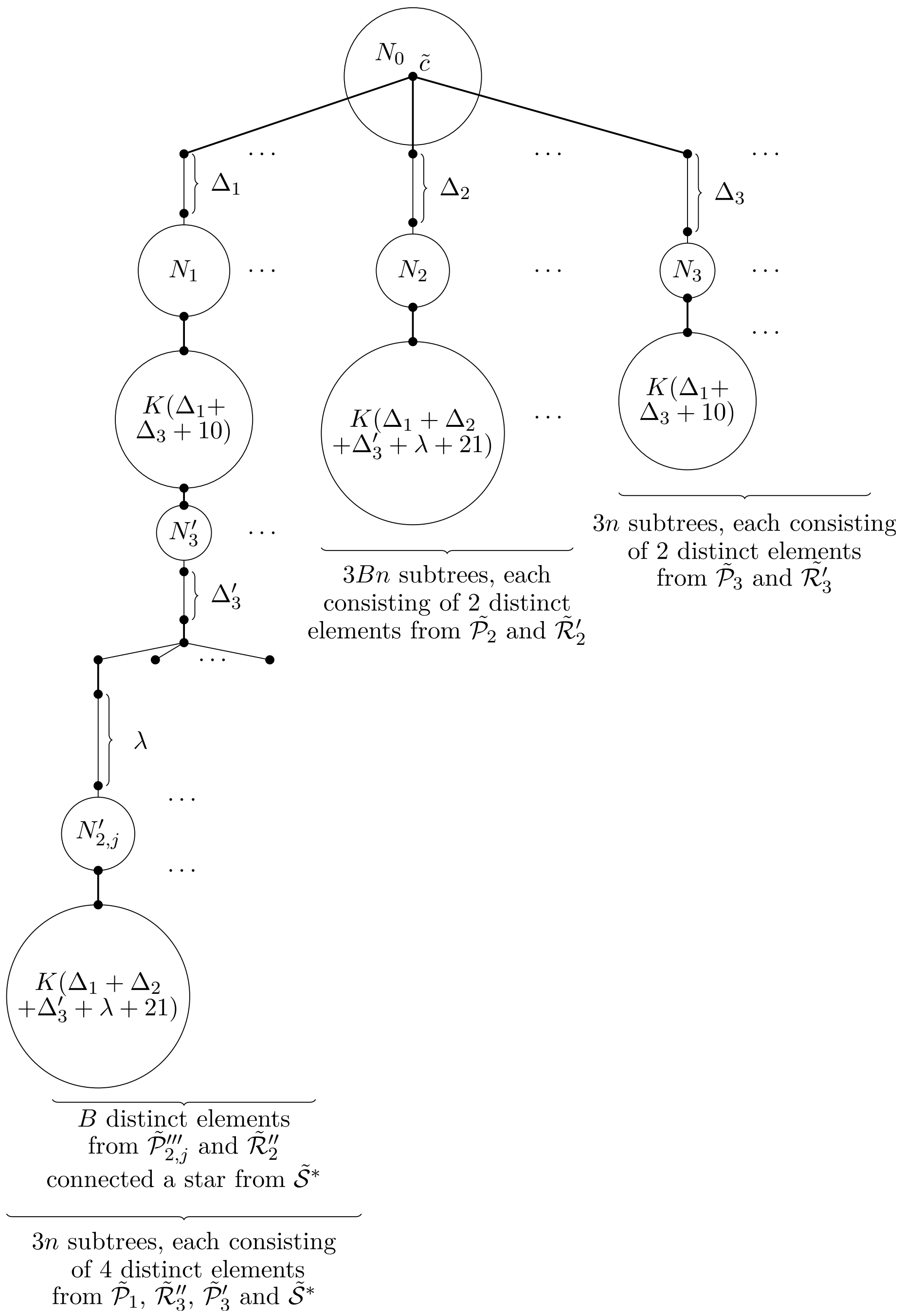

N_0
$\tilde{c}$
Δ_1
Δ_2
Δ_3
N_1
N_2
N_3
$K(\Delta_1 + \Delta_3 + 10)$
$K(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$
$K(\Delta_1 + \Delta_3 + 10)$
N_3'
Δ_3'
3Bn subtrees, each consisting of 2 distinct elements from $\tilde{\mathcal{P}}_2$ and $\tilde{\mathcal{R}}_2'$
3n subtrees, each consisting of 2 distinct elements from $\tilde{\mathcal{P}}_3$ and $\tilde{\mathcal{R}}_3'$
λ
$N_{2,j}'$
$K(\Delta_1 + \Delta_2 + \Delta_3' + \lambda + 21)$
B distinct elements from $\tilde{\mathcal{P}}_{2,j}'''$ and $\tilde{\mathcal{R}}_2''$ connected a star from $\tilde{\mathcal{S}}^*$
3n subtrees, each consisting of 4 distinct elements from $\tilde{\mathcal{P}}_1$, $\tilde{\mathcal{R}}_3''$, $\tilde{\mathcal{P}}_3'$ and $\tilde{\mathcal{S}}^*$

Fig. 5.7: The schematic representation of T_2 of CPM on trees with $l = 3$ and $d = 0$ corresponding to an instance of 3-PARTITION. Circles correspond to the stars. Thick lines correspond to the edges that connect components of T_2.

remaining leaves by $\tilde{\Lambda}''_j$. For $i = 1, \ldots, 3$, a_{j_i} many donor paths from $\mathcal{P}''_{2,j}$ are mapped to the recipient paths from $\tilde{\mathcal{P}}'''_{2,j}$, whose tail vertices are adjacent to $\tilde{\Lambda}'_j$; the tail vertices of $B - a_{j_i}$ donor paths from $\mathcal{P}'_{2,j}$ are mapped to the leaves in $\tilde{\Lambda}''_j$, and the remaining vertices of the donor paths are mapped to the recipient paths from $\tilde{\mathcal{P}}'''_{2,j}$, that are adjacent to $\tilde{\Lambda}''_j$.

In the following we will show that such a mapping is a compactness-preserving mapping. Let $\Delta := \Delta_1 + \Delta_2 + \Delta'_3 + \lambda + 21$, $\rho \in \mathcal{D}_2$, ρ' be ρ's first component subtree. Let r_1 and r_6 be the root and the tail of ρ', respectively, $r_5 \in V(\rho)$ be the vertex adjacent to r_6, x be the leaf of the star s of the donor path that is adjacent to r_6 in T_1, and y be the center of s. Without providing detailed computation steps we have: $L'_{r_1} - L_{r_1} = 0$, $L'_{r_5} - L_{r_5} = 4 - \Delta$, $L'_{r_6} - L_{r_6} = 0$, $L'_x - L_x = -1$, $L'_y - L_y = -1$. For the remaining vertices $u_j \in \mathcal{L}_j(\rho')$, $j = 2, \ldots, 6$, we outline that: $L'_{u_2} - L_{u_2} = 8 - 2 \cdot \Delta$, $L'_{u_3} - L_{u_3} = 18 - 9 \cdot \Delta$, $L'_{u_4} - L_{u_4} = 6 - 2 \cdot \Delta$, $L'_{u_5} - L_{u_5} = 3 - \Delta$, $L'_{u_6} - L_{u_6} = -1$. That is, for every considered vertex v, we have $L'_v - L_v \leq 0 = d$. Since the two component subtrees of a donor tree have the same structure, a similar result can be shown for the vertices of ρ's second component subtree.

Given $\Delta := \Delta_1 + \Delta_3 + 10$ and $\rho \in \mathcal{D}_3$ the mapping from ρ to the two recipient stars from $\tilde{\mathcal{R}}'_3$ and $\tilde{\mathcal{R}}''_3$ is also a compactness-preserving mapping.

"$\Longleftarrow$": Let f be a compactness-preserving mapping from T_1 to T_2. The next claims follow directly from the sizes of 3-neighborhoods and the image distance constraints of the mapping.

Claim 31.1: The vertices of S_0 have to be mapped to the vertices of $\tilde{S}_0$ with $\tilde{c} = f(c)$.

Claim 31.2: The star and the path of a donor path in $\mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ have to be mapped to the star and the path of a recipient path in $\tilde{\mathcal{P}}_1 \cup \tilde{\mathcal{P}}_2 \cup \tilde{\mathcal{P}}_3$, respectively.

Claim 31.3: The center of the star of a donor path in $\mathcal{P}'_{2,j} \cup \mathcal{P}''_{2,j}$ has to be mapped to the center of the star of a recipient path in $\tilde{\mathcal{P}}'''_{2,j}$, $j = 1, \ldots, n$.

Claim 31.4: The donor paths in $\mathcal{P}'_3$ have to be mapped to the corresponding recipient paths in $\tilde{\mathcal{P}}'_3$.

Let ρ' be the first component subtree of a donor tree $\rho \in \mathcal{D}_3$, π be the donor path in $\mathcal{P}_3$ that is connected to ρ', and $\tilde{\rho}'$ be the recipient star in $\tilde{\mathcal{R}}'_3$ that is adjacent to the path in $\tilde{\pi} \in \tilde{\mathcal{P}}_3$ with π mapped to $\tilde{\pi}$. Next, we show that the vertices in ρ' cannot be too far away from each other in T_2.

Claim 31.5: The vertices of ρ' have to be mapped to the vertices of $\tilde{\rho}'$.

Proof. The correctness of the claim is proved by a sequence of subclaims.

Let $\Delta := \Delta_1 + \Delta_3 + 10$ and r_1 and r_6 be the root and the tail of ρ', respectively; let r_i, $i = 2, \dots, 5$ be the remaining vertices lying on the path from r_1 to r_6 and $\tilde{r}$ be the tail of $\tilde{\rho}'$.
Subclaim 31.6: The tail r_6 has to be mapped to $\tilde{\rho}'$ with $\tilde{r} = f(r_6)$.
Subclaim 31.7: Every vertex from $N^3_{T_1}(r_6) \cap V(\rho')$ has to be mapped to some vertex of $\tilde{\rho}'$.
Subclaim 31.8: Let $v \in N^3_{T_1}(r_6) \cap V(\rho')$. Every vertex from $N^3_{T_1}(v) \cap (\mathcal{L}_4(\rho') \cup \mathcal{L}_5(\rho') \cup \mathcal{L}_6(\rho'))$ has to be mapped to some vertex of $\tilde{\rho}'$.
Subclaim 31.9: The vertex r_2 has to be mapped to some vertex of $\tilde{\rho}'$.
Subclaim 31.10: The root r_1 of ρ' has to be mapped to some vertex of $\tilde{\rho}'$.
Thus, the claim is true. □

Claim 31.11: The vertices of ρ's second component subtree have to be mapped to a recipient star in $\tilde{\mathcal{R}}''_3$.
By Claims 31.5 and 31.11, we have:
Claim 31.12: Let $\rho \in \mathcal{D}_2$. The vertices of ρ's first component subtree have to be mapped to the vertices of the corresponding recipient star in $\tilde{\mathcal{R}}'_3$, and the vertices of ρ's second component subtree have to be mapped to a star in $\tilde{\mathcal{R}}''_3$.
It remains to consider $\tilde{\mathcal{S}}^*$.
Claim 31.13: The stars in $\mathcal{S}^*$ have to be mapped to the corresponding stars in $\tilde{\mathcal{S}}^*$.

Now, by applying an induction from $j = n$ to $j = 1$, we prove that the number of leaves in the stars of $\mathcal{S}^*$ that are mapped to $\tilde{\mathcal{S}}^*_j \in \tilde{\mathcal{S}}^*$ is B for every j. For $j = n$, let $\tilde{s}_1, \tilde{s}_2, \tilde{s}_3$ denote the stars in $\tilde{\mathcal{S}}^*_j$ that are connected to $\tilde{\mathcal{P}}'''_{2,j}$; let $s_{j_1}, s_{j_2}, s_{j_3}$, where $1 \le j_1, j_2, j_3 \le 3n$, be the stars in $\mathcal{S}^*$ that correspond to $a_{j_1}, a_{j_2}, a_{j_3} \in A$ and are mapped to $\tilde{s}_1, \tilde{s}_2, \tilde{s}_3$, respectively.

Consider first $a_{j_1} + a_{j_2} + a_{j_3} > B$. Then there exists at least one donor path $\pi \in \mathcal{P}'_{2,n}$ that is not mapped to $\tilde{\mathcal{P}}'''_{2,n}$. Consequently, there is a vertex in T_2 with degree at least $N'_{2,n}$ that is an image of the center of the star in π. This is a contradiction, since all vertices with degree greater than $N'_{2,n}$ are images of star S_0 or the stars of the donor paths from $\mathcal{P}_1 \cup \mathcal{P}_2$.

Now consider $a_{j_1} + a_{j_2} + a_{j_3} < B$. Since $\sum_{a \in A} a = nB$, there exist $\tilde{s}_{j_4}, \tilde{s}_{j_5}, \tilde{s}_{j_6} \in \tilde{\mathcal{S}}^*$ that are connected to $\tilde{\mathcal{P}}'''_{2,j'}$, $1 \le j' \le n-1$, and there exist $s_{j_4}, s_{j_5}, s_{j_6} \in \mathcal{S}^*$ that correspond to $a_{j_4}, a_{j_5}, a_{j_6} \in S$, where $1 \le j_4, j_5, j_6 \le 3n$, and are mapped to $\tilde{s}_{j_4}, \tilde{s}_{j_5}, \tilde{s}_{j_6} \in \tilde{\mathcal{S}}^*$, such that $a_{j_4} + a_{j_5} + a_{j_6} > B$. Hence, there is at least one donor path in $\mathcal{P}'_{2,j'}$ that has to be mapped to a recipient path in $\tilde{\mathcal{P}}'''_{2,j''}$, $j' < j'' \le n-1$. Recursively, we come to a contradiction to the fact that the center of the star of a donor path in $\mathcal{P}'_{2,\ell}$, $\ell \le n-1$, can only be mapped to the center of the star of a recipient path in $\tilde{\mathcal{P}}'''_{2,n}$, which is in turn already mapped to the center of the star of a donor path from $\mathcal{P}'_{2,n}$. Thus, if the mapping f exists, then $a_{j_1} + a_{j_2} + a_{j_3} = B$. In a similar way, the same can be shown for $j = n-1$

to $j = 1$ and the remaining sets $\tilde{\mathcal{P}}'''_{2,j}$. The reason is that for any $j = 1, \ldots, n$ we have exactly $3B$ donor paths in $\mathcal{P}'_{2,j} \cup \mathcal{P}''_{2,j}$ which can only be mapped to the $3B$ recipient paths of $\tilde{\mathcal{P}}'''_{2,j}$. Moreover, each of the B donor paths in $\mathcal{P}''_{2,j}$ has exactly one vertex less than a donor path from $\mathcal{P}'_{2,j}$. Further, this implies that every three stars from $\mathcal{S}^*$, that are mapped to the three stars in $\tilde{\mathcal{S}}^*_j$, must have exactly B leaves in total. Thus, given the mapping f, we can derive the corresponding 3-partition for the instance (A, B). □

Applying edge subdivisions to the gadgets in the proof of Theorem 31 leads to reductions proving NP-hardness of CPM on trees with $l \geq 3$ and $d \geq 0$.

5.2 Polynomial-Time Solvable Cases

5.2.1 Case $l = 1$, $d = 0$

In the case of CPM on trees with $l = 1$ and $d = 0$ the problem is equivalent to the tree isomorphism problem, which can be solved in polynomial time [59].

Theorem 32. *CPM on trees with $l = 1$, $d = 0$ can be solved in polynomial time.*

Proof. Assume that there exists a compactness-preserving mapping f from T_1 to T_2 with $l = 1$ and $d = 0$. Let $v \in V_1$, and $f(v) \in V_2$.

We first show that for every $u \in N_{T_1}(v)$, $f(u) \in N_{T_2}(f(v))$. Assume that this is not true. Then there exists $x \in N_{T_1}(v)$ with $f(x) \notin N_{T_2}(f(v))$. Let $\Delta := \mathrm{dist}_{T_2}(f(v), f(x))$. Then, $\Delta \geq 2$, $L'_v \geq (|N_{T_1}(v)| - 1) + \Delta$ and $L'_v - L_v \geq (|N_{T_1}(v)| - 1) + \Delta - |N_{T_1}(v)| = \Delta - 1$. Then, $L'_v - L_v \leq 0$ only if $\Delta \leq 1$, which is a contradiction to $\Delta \geq 2$. Thus, for all $u \in N_{T_1}(v)$, it holds $f(u) \in N_{T_2}(f(v))$, which implies $|N_{T_1}(v)| \leq |N_{T_2}(f(v))|$.

Assume $|N_{T_1}(v)| < |N_{T_2}(f(v))|$. Then there exists $t \in V_1$ with $|N_{T_2}(f(t))| < |N_{T_1}(t)|$. This implies that there exists $x \in N_{T_1}(t)$ with $(f(t), f(x)) \notin E_2$, contradicting the fact proven above.

Thus, a mapping f from T_1 to T_2 exists if and only if $(v, u) \in E_1$ and $(f(v), f(u)) \in E_2$, which is equivalent to the tree isomorphism problem. □

5.2.2 Case $l = 2, d = 0$

Lemma 33. *If there is a compactness-preserving mapping f with $l = 2$ and $d = 0$ from T_1 to T_2, then for every $v \in V_1$, $|N_{T_1}(v)| = |N_{T_2}(f(v))|$.*

Proof. Assume that there is a vertex $v \in V_1$ with $|N_{T_1}(v)| > |N_{T_2}(f(v))|$. Then, with $L_v = |N_{T_1}(v)| + 2|\hat{N}^2_{T_1}(v)|$ and $L'_v \geq |N_{T_2}(f(v))| + 2(|N^2_{T_1}(v)| - |N_{T_2}(f(v))|) = 2|N^2_{T_1}(v)| - |N_{T_2}(f(v))|$, we have $L'_v - L_v \geq 2|N^2_{T_1}(v)| - 2|\hat{N}^2_{T_1}(v)| -$

$|N_{T_1}(v)| - |N_{T_2}(f(v))| = |N_{T_1}(v)| - |N_{T_2}(f(v))| > 0 = d$, which is a contradiction to the compactness-preserving property of $f(v)$. Thus, $|N_{T_1}(v)| \leq |N_{T_2}(f(v))|$.

Now assume $|N_{T_1}(v)| < |N_{T_2}(f(v))|$. Since $\sum_{y \in V_1} |N_{T_1}(y)| = \sum_{y' \in V_2} |N_{T_2}(y')| = 2|E_1|$, there exists $x \in V_1$ with $|N_{T_1}(x)| > |N_{T_2}(f(x))|$, which contradicts to the previous conclusion. Thus, for each $v \in V_1$, $|N_{T_1}(v)| = |N_{T_2}(f(v))|$. □

Lemma 34. *If there is a compactness-preserving mapping f with $l = 2$ and $d = 0$ from T_1 to T_2, then for every $v \in V_1$ and $u \in N^2_{T_1}(v)$, $f(u) \in N^2_{T_2}(f(v))$.*

Proof. Assume that there exists $u \in N^2_{T_1}(v)$ for a vertex $v \in V_1$, such that $f(u) \notin N^2_{T_2}(f(v))$. Thus, $\Delta = \mathrm{dist}_{T_2}(f(v), f(u)) \geq 3$. For the vertex v, we have $L_v = |N_{T_1}(v)| + 2|\hat{N}^2_{T_1}(v)|$ and, assuming that $|N_{T_1}(v)|$ many vertices in $N^2_{T_1}(v)$ are mapped to $N_{T_2}(f(v))$ and the remaining vertices, except the vertex u, are mapped to $\hat{N}^2_{T_2}(f(v))$, $L'_v \geq |N_{T_2}(f(v))| + 2(|\hat{N}^2_{T_1}(v)| - 1) + \Delta$. Then, it holds $L'_v - L_v \geq |N_{T_2}(f(v))| + 2(|\hat{N}^2_{T_1}(v)| - 1) + \Delta - (|N_{T_1}(v)| + 2|\hat{N}^2_{T_1}(v)|) = |N_{T_2}(f(v))| - |N_{T_1}(v)| + \Delta - 2 = \Delta - 2 > 0 = d$, which is a contradiction to the compactness-preserving property of $f(v)$, since $\Delta \geq 3$. □

The following two lemmas follow directly from Lemmas 33-34.

Lemma 35. *If there is a compactness-preserving mapping f with $l = 2$ and $d = 0$ from T_1 to T_2, then for every $v \in V_1$, $|\hat{N}^2_{T_1}(v)| = |\hat{N}^2_{T_2}(f(v))|$.*

Lemma 36. *If there is a compactness-preserving mapping f with $l = 2$ and $d = 0$ from T_1 to T_2, then for every $v \in V_1$, $u \in N_{T_1}(v)$, $\mathrm{dist}_{T_2}(f(v), f(u)) \leq 2$.*

Lemma 37. *Let T_1 be a tree with diameter greater than 3. If there is a compactness-preserving mapping f with $l = 2$ and $d = 0$ from T_1 to T_2, then for every $(v, u) \in E_1$, $(f(v), f(u)) \in E_2$.*

Proof. Assume the claim is not true. Let f be the compactness-preserving mapping. According to Lemma 33, the leaves from T_1 can only be mapped to the leaves in T_2. Let $x \in \mathrm{leaves}(T_1)$, $x' \in \mathrm{leaves}(T_2)$ with $x' = f(x)$. Suppose T_1 and T_2 are rooted such that x' has the maximal depth in T_2. Let u and u' be the parent vertices of x and x', respectively; v and v' be the parent vertices of u and u', respectively.

According to the above lemmas, the leaves from $\hat{N}^2_{T_1}(x)$ have to be mapped to the leaves within distance two from x'. The only non-leaf vertices in $N^2_{T_2}(x')$ are u' and v'. If u is mapped to v', then $|N_{T_1}(u)| = |N_{T_2}(v')|$ and v can only be mapped to u'. Then, all vertices in $N^2_{T_1}(v) \setminus N_{T_1}(u)$ have to be mapped either to the leaf children of u' or to the vertices in $N_{T_2}(v')$. Since the diameter of T_1 is greater than 3, the former one is not possible. For the latter one, since all leaf children of u' can only be mapped to the children of u and $|N_{T_1}(v)| = |N_{T_2}(u')|$, the children of u must be all mapped to the children of u'. This means $|N_{T_1}(u)| = |N_{T_2}(u')|$ and $|N_{T_1}(v)| = |N_{T_2}(v')|$, which excludes the second possibility. Therefore, $f(u) = u'$. Thus, the vertex u can only be mapped to u', i.e. $u' = f(u)$. If the vertex v is not mapped to v', then $\mathrm{dist}_{T_2}(x', f(v)) > 2$, contradicting to Lemma 36. Hence, it holds $v' = f(v)$.

Thus, the resulting so far mapping f keeps u and its neighbors adjacent in T_2, i.e. for every $t \in N_{T_1}(u)$, it holds $(f(t), f(u)) \in E_2$. Further, every $q \in \hat{N}^2_{T_1}(u)$ has to be mapped to $N_{T_2}(v') \setminus \{u'\}$; also note that $|\hat{N}^2_{T_1}(u)| = |N_{T_2}(v') \setminus \{u'\}|$. Thus, for $(q, v) \in E_1$, we have $(f(q), f(v)) \in E_2$. The similar has to be satisfied for $\hat{N}^2_{T_1}(v)$, which is mapped to $\hat{N}^2_{T_1}(v')$, and, continuing, for the remaining vertices in T_1 and T_2. In the conclusion, for every edge $(t_1, t_2) \in E_1$, we have $(f(t_1), f(t_2)) \in E_2$. □

Theorem 38. *CPM on trees with $l = 2$, $d = 0$ can be solved in polynomial time.*

Proof. By Lemma 37, if the diameter of T_1 is greater than 3, then the problem is equivalent to the tree isomorphism problem. We consider now the cases of diameter being 2 and 3. If T_1 is a star, then, by Lemma 33, T_2 must also be a star. If the diameter of T_1 is equal to 3, then there is a path in T_1 with 4 vertices x, u, v, t and all other vertices in T_1 are leaves and adjacent to u or v. Assume that the leaf x is mapped to a leaf $x' \in T_2$. Then, there are two non-leaf vertices $u', v' \in N^2_{T_2}(x')$ with $(x', u') \in E_2$. Obviously, the leaves adjacent to u have to be mapped to the leaves adjacent to u'. Consider first, u is mapped to u'; then v has to be mapped to v' and the leaves adjacent to v must be mapped to the leaves adjacent to v'. Thus, the diameter of T_2 is also equal to 3 and $|N_{T_1}(u)| = |N_{T_2}(u')|$, $|N_{T_1}(v)| = |N_{T_2}(v')|$. Consider now, u is mapped to v', then v has to be mapped to u'. By Lemma 33, $|N_{T_2}(v')| = |N_{T_1}(u)|$ and $|N_{T_2}(u')| = |N_{T_1}(v)|$. By Lemma 35, we have $|\hat{N}^2_{T_2}(x')| = |\hat{N}^2_{T_1}(x)|$, which implies that $|N_{T_2}(u')| = |N_{T_1}(u)|$ and that the leaves adjacent to v can only be mapped to the leaves adjacent to v'. Thus, in this case the diameter of T_2 is also equal to 3 and non-leaf vertices of both trees have the same degrees. This is clearly checkable in polynomial time. □

5.2.3 Case $l = 1, d = 1$

We need the following observations.

Lemma 39. *Let $v, u \in V_1$ with $(v, u) \in E_1$. If there is a compactness-preserving mapping f from T_1 to T_2, then* $\mathrm{dist}_{T_2}(f(v), f(u)) \leq 2$.

Proof. Assume that there exist $v \in V_1$ and $u \in N_{T_1}(v)$ such that $\Delta := \mathrm{dist}_{T_2}(f(v), f(u)) \geq 3$. We have $L'_v \geq (|N_{T_1}(v)| - 1) + \Delta$, and $L'_v - L_v \geq (|N_{T_1}(v)| - 1) + \Delta - |N_{T_1}(v)| = \Delta - 1 > 1$. □

Lemma 40. *Let $v \in V_1$ and $p := |\{u \in N_{T_1}(v) : \mathrm{dist}_{T_2}(f(v), f(u)) = 2\}|$. If there is a compactness-preserving mapping f from T_1 to T_2, then $p \leq 1$.*

Proof. Assume $p \geq 2$. We have $L'_v \geq (|N_{T_1}(v)| - p) + 2p$, and $L'_v - L_v \geq (|N_{T_1}(v)| - p) + 2p - |N_{T_1}(v)| = p > 1$. $\square$

Lemma 41. *Let $v, u, x \in V_1$ with $(v, u) \in E_1$, $(u, x) \in E_1$, and $(f(v), f(u)) \in E_2$. If there is a compactness-preserving mapping f from T_1 to T_2, then either for every $y \in V(T_1(x, u))$, $f(y) \in V(T_2(f(u), f(v)))$ or for every $y \in V(T_1(x, u))$, $f(y) \in V(T_2(f(v), f(u)))$.*

Lemma 42. *If there is a compactness-preserving mapping f from T_1 to T_2, then for every $v \in V_1$, $|N_{T_2}(f(v))| + 1 \geq |N_{T_1}(v)|$.*

In the following we assume that T_2 is rooted at an arbitrary vertex r' and tree T_1 remains unrooted.

Our algorithm firstly works from the leaves of T_2 to its root. At each vertex $v' \in V_2$, we iterate over all vertices $v \in V_1$ and decide whether v can be mapped to v'. To make this decision, we distinguish several cases, based on whether there is a neighbor u of v, such that one vertex $z \in T(u, v)$ could be mapped to the parent t' of v' as depicted in Fig. 5.8. Let $u_2 \in N_{T_1}(v) \setminus \{u\}$, u'_1, u'_2, u'_3 be three children of v', and $s' \in N_{T_2}(t') \setminus \{v'\}$. We know by Lemmas 40-42 $z \in N^3_{T_1}(v)$. In the first case, we assume that there is a $u \in N_{T_1}(v)$ mapped to t'. Then, at most one neighbor x of u can be mapped to a child u'_1 of v'. If vertex x does not exist, we say that x and u'_1 are null elements, denoted by $\emptyset$. In the second case, we assume that there is a vertex $z \in T(u, v)$ mapped to t' and $z \in \hat{N}^2_{T_1}(v)$, i.e. $z \in N_{T_1}(u) \setminus \{v\}$. Then u has to be mapped to either s' or u'_1. If u is mapped to s', then there is at most one neighbor y of x that can be mapped to a child u'_1 of v' (Fig. 5.8, (d)). In the third case, we assume $z \in T(u, v)$ and $z \in \hat{N}^3_{T_1}(v)$, i.e. z is a neighbor of $x \in N_{T_1}(u) \cap V(T(u, v))$ Then, either u is mapped to a child x'_1 of u'_1 and x is mapped to u'_1 or u is mapped to s' and x is mapped to $t'' \in N_{T_2}(s')$. For the first and second cases, where $z \in N^2_{T_1}(v)$ and u mapped to $N_{T_2}(v')$, there exists at most one neighbor u_2 of v that can be mapped to s' or to a child x'_2 of u'_2. If there is a neighbor u_2 of v mapped to x'_2, then, there exist at most one neighbor x_2 of u_2 and at most one neighbor y_2 of x_2 that can be mapped to two children u'_2 and u'_3 of v' (Fig. 5.8, (a) and (c)), respectively. Note if v' is the root r' of T_2, then z is a null element.

Given v and v', for every possible z, we guess all possible combinations of the vertices u, x, y, u_2, x_2, y_2 and $u'_1, u'_2, u'_3, x'_1, s', t''$ corresponding to the above cases to decide whether there can exist a compactness-preserving mapping f with $v' = f(v)$.

In the algorithm we encode the above combinations with configurations. A *configuration* $C = (c_1, c_2, c_3, c_4, c_5, c_6)$ means $c_1, c_3, c_5 \in V_1$ can be mapped to $c_2, c_4, c_6 \in V_2$, respectively, where $c_3 \in N^3_{T_1}(c_1)$, $c_5 \in N_{T_1}(c_1)$, c_2 is a child of c_4 and $\mathrm{dist}_{T_2}(c_2, c_6) = 2$. Given a configuration C, the algorithm computes two sets $\bar{N}_1$ and $\bar{N}_2$ that contain neighbors of v and v', respectively, that can

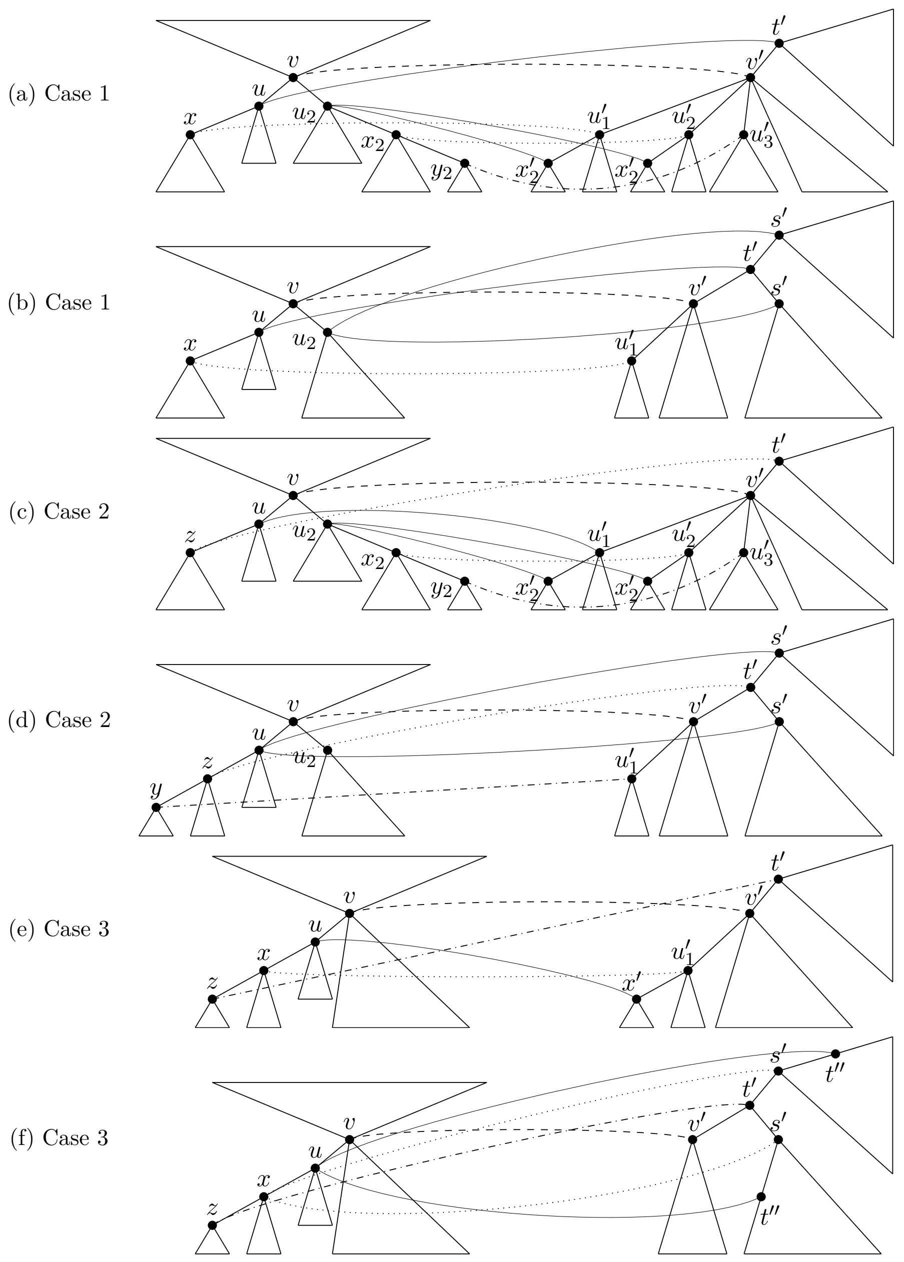


Fig. 5.8: The three cases of mapping $v \in V_1$ to $v' \in V_2$

be mapped to the configuration C. That is, by the definition of the cases, we have $\bar{N}_1 := \{u, u_2\}$ and $\bar{N}_2 := \{t', u'_1, u'_2, u'_3\}$. Notice that depending on the configuration under consideration some of the vertices in $\bar{N}_1$, $\bar{N}_2$ can be null elements.

For v, v' and configuration C, let $\mathcal{N}_1 := N_{T_1}(v) \setminus \bar{N}_1$ and $\mathcal{N}_2 := N_{T_2}(v') \setminus \bar{N}_2$ We say that there exists a matching $F(C)$ for a configuration C, if there are a set $B \subseteq \mathcal{N}_1 \cup N_{T_1}(\mathcal{N}_1) \setminus \{v\}$ and a bijection g between B and $\mathcal{N}_2$, such that the subtree $T_1(x)$ for each $x \in B$ can be mapped to the subtree $T_2(g(x))$. For a vertex $x \in B$, if $N_{T_1}(x) \cap B = \emptyset$, then $T_1(x) := T_1(x, v)$. Otherwise, let y be the only element in $N_{T_1}(x) \cap B$; in the case of $x \in N_{T_1}(v)$, $T_1(x) := T_1(x, v) \setminus T_1(y, x)$; in the case of $y \in N_{T_1}(v)$, $T_1(x) := T_1(x, y)$. A configuration C is *realizable* if there exists such a matching $F(C)$. A *confirming configuration* of a configuration C, denoted by $\hat{C}$, is a configuration $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4, \hat{c}_5, \hat{c}_6)$ with $\hat{c}_3 = c_3$, $\hat{c}_4 = c_4$, $\hat{c}_1 = c_5$, $\hat{c}_2 = c_6$, $\hat{c}_5 = c_1$, $\hat{c}_6 = c_2$.

The algorithm. The algorithm works in two phases. In the first phase we process bottom-up and, for every vertex $v' \in V_2$, we compute the set of all realizable configurations by combining the sets of realizable configurations of the children of v'. We check for a configuration C^* whether there exists a matching F^*. If yes, then it saves C^* together with F^* for vertex v'; otherwise, it discards C^*.

In the second phase, given that there is at least one realizable configuration at the root of T_2, we construct the actual compactness-preserving mapping f in a top-down manner. Let $\mathcal{C}$ be the set of realizable configurations of all vertices. Note that, if the set of realizable configurations at a vertex $v' \in V_2$ is empty, then no compactness-preserving mapping exists and the algorithm returns "no".

Phase 1. We first compute realizable configurations for the leaves of T_2. Let v' be a leaf of T_2. According to Lemma 42, v' can only be mapped to a vertex $v \in V_1$ with degree at most 2. Consider first v is a leaf. Given the only neighbor u of v, we only need to test configurations $(v, v', u, t', \emptyset, \emptyset)$ and (v, v', x, t', u, s'), where $x \in N_{T_1}(u) \setminus \{v\}$ and $s' \in N_{T_2}(t') \setminus \{v'\}$. Now let $v \in V_1$ be of degree two and u_1, u_2 are the only neighbors of v. We test configurations $(v, v', u_i, t', u_{3-i}, s')$, where $i = \{1, 2\}$ and $s' \in N_{T_2}(t') \setminus \{v'\}$.

Consider now a non-leaf vertex $v' \in V_2$ and a vertex $v \in V_1$. Let $\mathcal{C}_{v,v'} := \{C \in \mathcal{C} : c_4 = v', v = c_3\}$. Given v is mapped to v', we combine sets of realizable configurations of the children of v' to compute realizable configurations for v'. For v and v' we generate every possible configuration C^* according to the described cases and test if C^* is realizable.

Next, we describe how to check whether a configuration C^* is realizable. First, we compute $\bar{N}_1$, $\bar{N}_2$ and a set M, which contains all configurations for the vertices in $N^2_{T_1}(v) \setminus \bar{N}_1$ specifying how they should be mapped to $\mathcal{N}_2$ given that

C^* is realizable. The set M is defined as $M := \{C \in \mathcal{C}_{v,v'} : c_1 \in \mathcal{N}_1, c_6 \notin \bar{N}_2$, and if $c_6 \in N_{T_2}(v')$, then there exists a confirming configuration $\hat{C}$ for $C \in \mathcal{C}_{v,v'}\}$. Note that a configuration $C \in \mathcal{C}_{v,v'}$ satisfying $c_6 \in N_{T_2}(v')$ is generated by assuming that there is a vertex $c_5 \in N_{T_1}(c_1)$ such that the subtree $T_1(c_5, c_1)$ is completely mapped to the subtree $T_2(c_6)$, where c_6 is a child of $c_4 = v'$. Now, while moving from c_2 to its parent v', we have to verify whether this assumption is sound to create realizable configurations for v'. This can be done by checking the existence of the confirming configurations for C.

The assumptions of all cases require that the vertices from $\mathcal{N}_1$ have to be mapped to the vertices in $\mathcal{N}_2$. Moreover, depending on the configurations in M, for every $u \in \mathcal{N}_1$, there could be at most one vertex in $N_{T_1}(u) \setminus \{v\}$, which is mapped to $\mathcal{N}_2$ as well.

With the help of M, we can compute a matching for C^*. Let $u_1, \ldots, u_m$ be the neighbors of v for which there is a configuration $C \in M$ with $c_1 = u_i$ for $1 \leq i \leq m$ and let $M_i := \{C \in M : c_1 = u_i\}$ for $i = 1, \ldots, m$. Given the definitions of configuration C^* and the corresponding set M, C^* is realizable if there is a "matching" set M^* of configurations $C^1, \ldots, C^m$ with $C^i \in M_i$ such that (1) $m = |\mathcal{N}_1|$, (2) $\{c_2, c_6 : C^i \in M^*\} \cap \{c_2, c_6 : C^j \in M^*\} = \emptyset$ for $i \neq j$, and (3) $\{c_2, c_6 : C \in M^*\} = \mathcal{N}_2$.

In the following we construct a weighted bipartite graph $B = B_1 \cup B_2$ based on the sets $M_1, \ldots, M_m$. For every configuration $C \in M_i$, we add a corresponding vertex $\hat{q}_i^j$, $j = 1, \ldots, |M_i|$ to B_1. For every vertex $u_i' \in \mathcal{N}_2$ we add to B_2 the corresponding vertex q_i. Additionally we add $2 \cdot (|M_i| - 1)$ vertices p_i^j, $j = 1, \ldots, 2 \cdot (|M_i| - 1)$ to B_2. Let $w_0 := 1$ and $w_i := 1 + 2n \cdot \sum_{\ell=0}^{i-1} w_\ell$. Further, we add an edge with weight w_i between every $\hat{q}_i^{j_1}$ and every $p_i^{j_2}$, $j_1 = 1, \ldots, |M_i|$, $j_2 = 1, \ldots, 2 \cdot (|M_i| - 1)$. For every $\hat{q}_i^j$ that corresponds to configuration $C \in M_i$, we add an edge with weight 1 between $\hat{q}_i^j$ and the vertex $p' \in B_2$ that corresponds to c_2 of C. Further, if $c_6 = \emptyset$, then we add an additional "dummy" vertex and connect it to $\hat{q}_i^j$ by adding an edge with weight 1; otherwise we add an edge with weight 1 between $\hat{q}_i^j$ and the vertex $p'' \in B_2$ that corresponds to c_6. The construction of B is completed with applying the above procedure to every $i = 1, \ldots, m$.

Now, we compute the matching set M^* by solving the so-called Constrained Weighted P_2-Packing on Bipartite Graphs (CWPB) problem on B, where given a weighted bipartite graph $B = (B_1 \cup B_2, E_{12})$ in which each edge is associated with a positive weight, and an integer q, we seek for a P_2-packing $\mathcal{P}$ of size q with end-vertices in B_2 such that the weight of $\mathcal{P}$ is maximum over all such P_2-packings. CWPB can be solved in $O(q \cdot (|E_{12}| + |V(B)| \cdot \log|V(B)|))$ time [60]. The correctness of the following lemma follows from the construction of the graph B and the weights of its edges.

Lemma 43. *The configuration C^* is realizable if and only if there is a P_2-packing for the instance $(B, |M|)$ of CWPB.*

Given an output $\mathcal{P}$ for $(B, |M|)$, we derive the corresponding matching set M^* with the dummy vertices in B representing empty sets. Consequently, we save the matching $F(C^*)$ corresponding to M^* together with C^*.

Phase 2. Given a realizable configuration C at root r' of T_2, we first construct a compactness-preserving mapping f for r' and its children. We assign $r' = f(c_1)$, $c_5 = f(c_6)$ and use assignments from the matching $F(C)$. Then, we proceed top-down and construct f for children of every assigned vertex v'. Consider a realizable configuration $b = (f^{-1}(v'), v', f^{-1}(t'), t', b_3, b_3')$. If $b_3' \in T_2 \setminus (T_2(v') \cup \{t'\})$, then we use $F(b)$ to construct f for the children of v'; otherwise, we build f for the children of v' with $b_3 = f(b_3')$. If a child u' of v' is already assigned with $u = f^{-1}(u')$, then for the children of v' we retrieve a matching $F(b)$ with an additional requirement that u is matched to u'.

Theorem 44. *CPM on trees with $l = 1$, $d = 1$ can be solved in polynomial time.*

Proof. In both phases we iterate over vertices $v' \in V_2$. The vertex v' can have at most $|V_1|$ vertices that can possibly be mapped to v'. Then, for every pair v' and v, we generate all possible configurations that requires at most $O(n^6)$ combinations. In order to check if a configuration is realizable, we construct graph B for CWPB, whose size is clearly polynomial in n. Thus, the overall running time is polynomial. □

5.2.4 Case $l = 2$, $d = 1$

For the case of CPM with $l = 2$ and $d = 1$, we can observe similar properties as Lemmas 39-42. In the following we only provide lemmas omitting their proofs.

Lemma 45. *Let $u \in V_1$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$, then $|N_{T_2}(f(u))| \geq |N_{T_1}(u)| - 1$.*

Lemma 46. *Let $u \in V_1$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$, and $|N_{T_2}(f(u))| = |N_{T_1}(u)| - 1$, then for every $z \in N^2_{T_1}(u)$ it holds $f(z) \in N^2_{T_2}(f(v))$.*

Lemma 47. *Let $u \in V_1$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$, and $|N_{T_2}(f(u))| = |N_{T_1}(u)| - 1$, then either (see Fig. 5.9):*

1. *$N_{T_1}(u) = \{v, x_1, x_2\}$ and $x_1, x_2 \in \text{leaves}(T_1)$, $N_{T_2}(f(u)) = \{f(x_2), f(v)\}$ and $f(x_1) \in N_{T_2}(f(v))$, or*

2. *$N_{T_1}(u) = \{v, x_1\}$ and $x_1 \in \text{leaves}(T_1)$, $N_{T_2}(f(u)) = \{f(v)\}$ and $f(x_1) \in N_{T_2}(f(v))$, or*

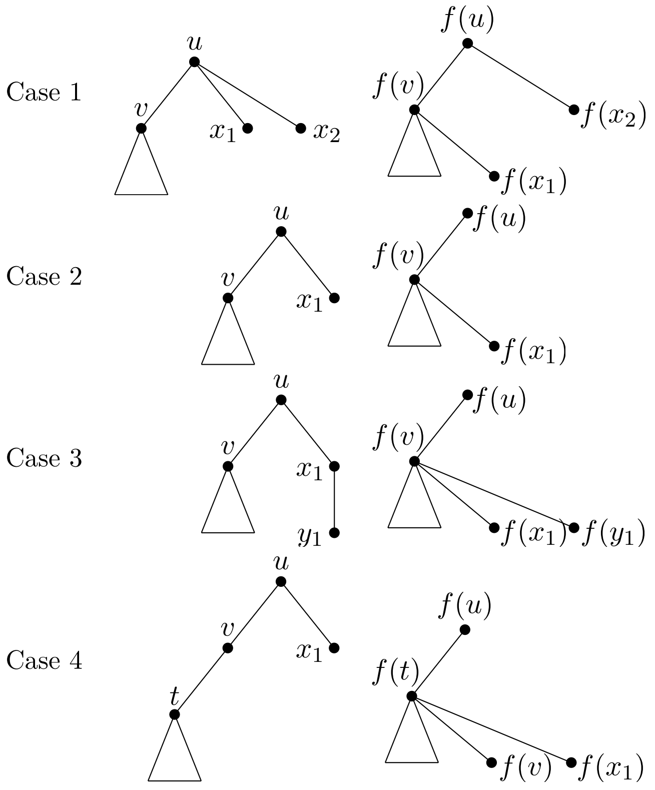


Fig. 5.9: The four cases of mapping $u \in V_1$ to V_2 in Lemma 47

3. $N_{T_1}(u) = \{v, x_1\}$, $N_{T_1}(x_1) = \{u, y_1\}$, *and* $y_1 \in \text{leaves}(T_1)$, $f(u), f(x_1)$, $f(y_1) \in N_{T_2}(f(v))$, *or*

4. $N_{T_1}(u) = \{v, x_1\}$, $N_{T_1}(v) = \{t, u\}$, *and* $x_1 \in \text{leaves}(T_1)$, $f(u), f(x_1)$, $f(v) \in N_{T_2}(f(t))$.

For the cases 1-3 in Lemma 47, the image of only one vertex from $N_{T_1}(u)$ can be adjacent to $f(v)$ (the similar is for vertex t in Case 4). Thus, with $\sum_{z \in V_1} |N_{T_1}(z)| = \sum_{z' \in V_2} |N_{T_2}(z')|$ for any trees T_1 and T_2, it holds the following:

Lemma 48. *Let* $u \in V_1$. *If there is a compactness-preserving mapping* f *with* $l = 2$ *and* $d = 1$, *and* $|N_{T_1}(u)| < |N_{T_2}(f(u))|$, *then there exist* $\mathcal{N} \subset N_{T_1}(u)$, *such that for every* $z \in \mathcal{N}$ *and* $\tilde{\mathcal{N}} := |V(T_1(z, u))|$ *it holds* $\tilde{\mathcal{N}} \leq 3$ *and for every* $\tilde{z} \in \tilde{\mathcal{N}}$, $f(\tilde{z}) \in N_{T_2}(f(u))$.

Lemma 49. *Let* $u \in V_1$ *and* $v \in N_{T_1}(u)$. *If there is a compactness-preserving mapping* f *with* $l = 2$ *and* $d = 1$, *then* $\text{dist}_{T_2}(f(u), f(v)) \leq 2$.

Lemma 50. *Let $u \in V_1$ and $z \in \hat{N}^2_{T_1}(u)$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$, then $\mathrm{dist}_{T_2}(f(u), f(z)) \leq 3$.*

Lemma 51. *Let $u \in V_1$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$, then for every $z \in N_{T_2}(f(u))$, it holds $f^{-1}(z) \in N^3_{T_1}(u)$.*

Lemma 52. *Let $u \in V_1$ and $v \in N_{T_1}(u)$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$ such that $\mathrm{dist}_{T_2}(f(u), f(v)) = 2$ and there is $z \in \hat{N}^2_{T_1}(u) \setminus N_{T_1}(v)$ with $f(u), f(v) \in N_{T_2}(f(z))$, then $v \in \mathrm{leaves}(T_1)$, $N_{T_1}(u) = \{x, v\}$, $N_{T_1}(x) = \{z, u\}$ and $f(x) \in N_{T_1}(z)$ (compare to Case 3 in Fig. 5.9).*

Lemma 53. *Let $u \in V_1$ and $v \in N_{T_1}(u)$. If there is a compactness-preserving mapping f with $l = 2$ and $d = 1$ such that $\mathrm{dist}_{T_2}(f(u), f(v)) = 2$, and there exists $x \in N_{T_1}(u) \setminus \{v\}$ with $f(u), f(v) \in N_{T_2}(f(x))$, then there is at most one vertex $\tilde{y} \in N_{T_1}(x) \setminus \{u\}$ and at most one vertex $\tilde{x} \in N_{T_1}(u) \setminus \{x, v\}$ such that, given $\mathcal{N}_1 := N_{T_1}(u) \setminus \{x, v, \tilde{x}\}$ and $\mathcal{N}_2 := N_{T_1}(x) \setminus \{u, \tilde{y}\}$, for every $z_1 \in \mathcal{N}_1$ and every $z_2 \in \mathcal{N}_2$ it holds $f(z_1) \in N_{T_2}(f(x))$, $f(z_2) \in N_{T_2}(f(u))$, $f(\tilde{x}) \in N_{T_2}(f(u))$, $f(\tilde{y}) \in N_{T_2}(f(x))$ and $|\mathcal{N}_1| - |\mathcal{N}_2| \leq 2$.*

In the above lemma the vertices u and x are "swapped": Almost all neighbors of x are mapped to the neighborhood of $f(u)$, and almost all neighbors of u become adjacent to $f(x)$.

Given the above lemmas, the basic ideas of the algorithm for CPM on trees with $l = 1$, $d = 1$ could also apply to derive an algorithm for CPM trees with $l = 2$, $d = 1$.

Theorem 54. *CPM on trees with $l = 2$, $d = 1$ can be solved in polynomial time.*

5.3 ILP Formulation of CPM with Isolation Set

Similarly to NPM, the optimization version of CPM asks to minimize the size k of the isolation set. In the presented ILP formulation of the problem we are looking for a mapping between vertices of graphs. In the mapping isolated vertices are not mapped to any other vertices (remember that an isolated vertex can cover any vertex in a graph). Instead of minimizing the number isolated vertices, we maximize the number of mapped vertices. Then, CPM can be presented as follows:

$$\text{maximize} \sum_{\substack{t \in V(G_1),\\ w \in V(G_2)}} x_{tw} \ , \tag{5.1}$$

$$\text{subject to} \qquad x_{tw} = \{0,1\} \ , \qquad \forall\, t \in V(G_1),\ \forall\, w \in V(G_2)$$

$$\sum_{w \in V(G_2)} x_{tw} \leq 1 \ , \qquad \forall\, t \in V(G_1) \tag{5.2}$$

$$\sum_{t \in V(G_1)} x_{tw} \leq 1 \ , \qquad \forall\, w \in V(G_2) \tag{5.3}$$

$$\begin{aligned} &(x_{tv} - 1) \cdot |V(G_2)|^2 \\ &\quad + \sum_{u \in N^l_{G_1}(t)} \left(\sum_{w \in V(G_2)} \mathrm{dist}_{G_2}(v,w) \cdot x_{uw} \right) \\ &\quad - \sum_{u \in N^l_{G_1}(t)} \left(\mathrm{dist}_{G_1}(t,u) \cdot \sum_{w \in V(G_2)} x_{uw} \right) \leq d \ , \\ &\qquad\qquad \forall\, t \in V(G_1), v \in V(G_2) \end{aligned} \tag{5.4}$$

Theorem 55. *The above ILP formulation of the optimization version of CPM is correct.*

Proof. We set x_{tw} to 1 whenever vertex $t \in V(T_1)$ is matched to vertex $w \in V(T_2)$; otherwise x_{tw} is 0. Conditions (5.2) and (5.3) ensure that every vertex in T_1 is mapped to at most one vertex in T_2 and vice versa. In condition (5.4) the first term is equal to $|V(G_2)|^2$ (or other very large number) if vertex t was mapped to v; if t is isolated, then condition (5.4) is always satisfied. The expression in the braces of the second term is equal to 0 if vertex u is isolated; otherwise, it is equal to the distance from the image v of t to an image w of an l-neighbor u of t. Similarly, the expression in the braces of the third term is equal to 0 if vertex u is isolated; otherwise, it is equal to the distance from t to u. The external sum of these terms adds up the distances to all l-neighbors and its images. Thus, if t is mapped to v, the second and the third terms of condition (5.4) correspond to the image distance L'_t and the proper distance L_t, respectively. Hence, the condition (5.4) is satisfied if and only if the difference between L'_t and L_t is not greater than d. □

6 Discussions and Outlook

The notion of the topological preservation of an alignment is generally defined as follows: Given two high confidence networks, a good alignment should map neighboring vertices of the first graph so that their images are also close in the second graph. Introducing the problems studied in Chapters 3-5, we aimed to formalized this concept in multiple ways. Then, to shed some light on what makes these problems difficult, we studied the complexity of the problems with input graphs restricted to trees.

The Graph Edit Distance with edge Insertion and edge Deletions problem (GED-ID), whose restricted to trees version (TED-ID) we considered in Chapter 3, supports the notion of the topology preservation implicitly only partially. For similar graphs we expect the optimal solution to have a lower number of required edge insertions and deletions, and thus many matched edges. At the same time, some parts of the first graph may be mapped very far from each other since deleting edges decomposes the first graph into separate components. Especially for trees, to decompose graphs can be done by deleting just one edge. Note the same situation can be observed in Maximum Common Subgraph (MCS) that is related to GED-ID. In GED-ID, the optimal mapping for the first graph being mapped to the second is inverse to the optimal mapping of the second graph being mapped to the first graph and these two mapping will have the same sum of edge insertion and edge deletions. The same holds for the number of edges in the solution graph of MCS. In order to hinder decomposition of the first graph into separate subgraphs , we make use (Chapter 4) of an extended neighborhood denoted by parameter l in NEIGHBORHOOD-PRESERVING MAPPING (NPM). For larger values of l decomposing the first graph into separate components by isolating some vertices becomes more expensive. Further, parameter d in NPM relaxes the notion of edge preservation, which forces edges to be mapped to each other. By varying parameters d and l we can specify the degree of topology preservation. In NPM, even for relatively small values of d it is possible to map arbitrarily many vertices from the first within distance d in the second graph. In Chapter 5 we partially restrict NPM by introducing COMPACTNESS-PRESERVING MAPPING (CPM). CPM is orthogonal to NPM (with $k = 0$) since in order to map arbitrarily many close neighbors of a vertex far from its image in the second graph, the value of d in CPM has to be much bigger than in NPM. At the same time, in CPM by mapping some l-neighbors of a vertex closer to its image, it is possible to map the other l-neighbors of the vertex arbitrarily far from its image even if the parameter d is relatively small; for NPM, in this case the value of d would have to be very large. For larger values of l in NPM, non-isolated vertices adjacent to isolated ones still have to be close to each other in the second graph.

Obviously, given two graphs of the same size, the problems GED-ID with zero allowed edge insertion and deletion, MCS with the size of the input graphs, NPM with $l = 1$ and $d = 1$, and CPM with $l = 1$ and $d = 0$ are all equivalent to the graph isomorphism problem. On general input graphs however, they demonstrate different behavior.

Note that unlike GED-ID and MCS , which by their definitions are based on the global properties of the solution mapping, NPM and CPM are defined with the help of properties of every vertex and its image. Additionally, in contrast to GED-ID (as well as MCS), the definitions of NPM and CPM do not ask how different the input graph are, but rather how different the first graph is from the second. For given parameters l and d, the size k of the isolation set in NPM corresponds to the amount of intrusion that have to be made to map the first graph to the other. In CPM parameter d plays the similar role. Both, the solution mappings in NPM and CPM can substantially be influenced by the topology in the input graphs, and not the direct difference in number of edges. Regardless of the connections in the first graph, the constraints (the parameters d and k) in these problems should be easier to satisfy if the second graph is more connected clarify.

Consider now the first graph with one false edge only. GED-ID does not distinguish if this edge connects close or distant vertices: The cost of deleting the edge remains the same. In NPM and CPM, if the false edge is “local”, i.e., it connects close vertices whose images should also be close in the second graphs, the solution mappings should not be affected too much; if the false edge connects vertices whose images should actually be distant in the second graph, parameter d of NPM and CPM has to be very large for a mapping to exist. Thus, given an incomplete network to be mapped to a less incomplete network, applying NPM and CPM is reasonable to address local errors with smaller values of l and d.

Next, we define a set of further problems that represent both theoretical and practical interests. First, consider a variation of NPM without isolation set ($k = 0$). Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_1)$ with $|V_1| = |V_2|$, two integer l and d, and a one-to-one mapping $f : V_1 \to V_2$, for $v \in V_1$ let $d'_v := \max_{u \in N^l_{G_1}(v)} \{\text{dist}_{G_2}(f(v), f(u))\}$ and $k'_v := |\{u \in N^l_{G_1}(v) : \text{dist}_{G_2}(f(v), f(u)) >$

$d\}|$. Using definitions of d'_v and k'_v, we introduce the following four optimization problems that ask to find a one-to-one mapping f that minimizes:

- $\sum_{v \in V_1} d'_v$ (NPM-Sum-Distance, NPM-SD),
- $\max_{v \in V_1} d'_v$ (NPM-Max-Distance, NPM-MD),
- $\sum_{v \in V_1} k'_v$ (NPM-Sum-Isolations,NPM-SI),
- $\max_{v \in V_1} k'_v$ (NPM-Max-Isolations NPM-MI).

Further, we propose the following variations of CPM that ask to find a one-to-one mapping f that minimizes:

- $\sum_{v \in V_1} (L'_v - L_v)$ (CPM-Sum-Difference, CPM-SD),
- $\max_{v \in V_1} \{L'_v - L_v\}$ (CPM-Max-Difference, CPM-MD),
- $\sum_{v \in V_1} \max\{L'_v - L_v, 0\}$ (CPM-Sum-Difference-Null, CPM-SDN),
- $\max_{v \in V_1} \max\{L'_v - L_v, 0\}$ (CPM-Max-Difference-Null, CPM-SDN).

Note that in CPM the difference between the image and the proper distances accumulates the total error between a vertex $v \in V_1$ and its l-neighbors (for $l \geq 2$). However, this allows to map some of the neighbors closer to an image of v, while others can be at a distance greater than l, with the total error being smaller. It is interesting to study the change in the complexity between CPM and problems that restricts such compensation of the error or measures the absolute error between the proper and image distance: More formally, a variation of CPM (CPM without compensation) may ask if there is a one-to-one mapping $f : V_1 \rightarrow V_2$ such that for every $v \in V_1$, $\sum_{u \in N^l_{G_1}} \max\{\text{dist}_{G_2}(f(v), f(u)) - \text{dist}_{G_1}(v, u), 0\} \leq d$; the other variation of CPM (CPM with absolute error) may ask if there is a one-to-one mapping f such that for every vertex $v \in V_1$, $|L'_v - L_v| \leq d$.

As for practical applications, the definitions of TED-ID and CTS represent mainly theoretical interest. Despite the complexity of GED-ID, NPM, and CPM, they and their combinations still can be useful in modeling Network Alignment.Thus, they require development of efficient heuristics to compute approximate solutions. Moreover, the definitions of CPM and NPM allow to derive a large variety of absolute and relative measures to estimate topological quality of alignments.

!

Part III

Heuristics for Network Alignment Using Graph Edit Distance Models

7 Background and Related Work

7.1 Introduction

Given two protein-protein interaction (PPI) networks, Network Alignment aims to find a biologically meaningful correspondence between nodes of the input networks. A procedure of constructing an alignment usually uses topology of the input networks and additional biological information. Correspondingly, the quality Q of an alignment combines both topological and biological aspects. A definition of Q balances the importance of the topology over biology and is expressed as, for example, $Q := \alpha \cdot T + (1 - \alpha) \cdot B$, where T, B, and $\alpha \in [0, 1]$ correspond to the topological quality, biological quality, and balance between them, respectively. Naturally, if the topologies of two networks are identical then every interaction in the first network can be mapped to an interaction of the other network and vice versa. Commonly, the topological aspect T counts the number of aligned edges and is to maximize. While the topological aspect is straightforward to interpret, the biological aspect B, in contrast, depends on the data that are used aligning the networks and their interpretation. Since one can model contribution of various biological data differently, we will focus on Network Alignment using topological information only.

In this part of the thesis we describe several heuristics that address Network Alignment of Protein-Protein Interaction Networks. Aligning networks, we look for a node-to-node mapping (an alignment). Our models aim to minimize the graph edit distance GED' induced by the mapping. By minimizing GED' we indirectly maximize the number of aligned edges.

The rest of this chapter provides an overview of the related work. Then, in Chapter 8, we introduce necessary definitions and describe NABEECO, which is a novel and robust heuristic based on Bee Colony Optimization. On a set of protein-protein interaction networks we compare NABEECO to the current state-of-the-art tool for the pairwise global network alignment problem, MI-GRAAL.

Further, in Chapter 9 we describe another heuristic, called GEDEVO that outperforms NABEECO. GEDEVO is a novel evolutionary algorithm for Network Alignment aiming to minimize the GED. We compare our implementation of GEDEVO against a set of tools: SPINAL, GHOST, C-GRAAL, and MI-GRAAL. On a set of protein-protein interaction networks from different organisms we demonstrate that GEDEVO outperforms these methods. It thus refines the previously suggested alignments based on topological information only.

In Chapter 10, we provide a formalization of the global network alignment problem for multiple networks. Given a set of PPI networks for different species we might ask how much the network topology is conserved throughout evolution. We model this problem as Topological Multiple one-to-one Network Alignment (TMNA), where we aim to minimize the total graph edit distance induced by pairs of mappings between the input networks. We extend our approach used in GEDEVO and developed a software tool, GEDEVO-M, that is able to align multiple PPI networks using topological information only. We demonstrate the power of our approach by computing a common subnetwork for a set of bacterial and eukaryotic PPI networks.

Due to the underlying GED model, the proposed heuristics, in contrast to many other tools, can be applied to all kinds of networks and allows incorporating prior knowledge about node/edge similarities. The three tools, NABEECO, GEDEVO and GEDEVO-M as well as the used data sets are publicly available at `http://nabeeco.mpi-inf.mpg.de`, `http://gedevo.mpi-inf.mpg.de`, and `http://gedevo.mpi-inf.mpg.de/multiple-network-alignment/`.

7.2 Pairwise Network Alignment

The network alignment problem has been defined in two general ways: local and global. Both definitions have been addressed by a set of tools that exploit different approaches.

Given two networks, the *local* network alignment problem aims to identify parts of the input networks that expose topological or biological similarity. PathBLAST [61] was one of the first tools dealing with the problem. PathBLAST aligns two input networks by combining topology and protein sequence similarities in order to identify conserved interaction pathways and complexes. By integrating interactions and sequence information another tool called NetworkBLAST [62, 63] can output a set of putative complexes that are evolutionary conserved across the two networks. Mawish [64] is based on evolutionary models and extends concepts of match, mismatch, and duplication from sequence alignment to Network Alignment; it processes an edge-weighted graph to evaluate similarity between graph structures through a scoring function that accounts for these evolutionary events. Graemlin [65] is based on equivalence classes and combines progressive alignment and seed-and-extend methods adapted from sequence alignment.

In contrast to Local Network Alignment, Global Network Alignment aims to find a correspondence between all nodes of the input networks. A set of methods that follows this definition of Network Alignment has been proposed. To align two networks, IsoRank [67] uses similarities of neighborhoods and sequences of nodes and represents the problem as an eigenvalue problem. In [72] Network Alignment is reformulated as a graph matching problem and approximated on relaxations over the set of doubly stochastic matrices (methods PATH and GA). GRAAL [68] is one of first approaches that follow the "seed-and-extend" strategy applied to Global Network Alignment. H-GRAAL [69] uses the Hungarian algorithm to find the best mapping from a constructed bipartite graph. Note a heuristic that utilizes bipartite graph representation of the input networks and the Hungarian algorithm to resolve it was also proposed to approximate the cost in the related GED problem [73]. PISwap [74, 75] is based on a local optimization heuristic: It first identifies an optimal alignment based on protein sequence similarity and refines it by propagating topology information. NATALIE 2.0 [76] (see also [77]) models the alignment as a generalization of the quadratic assignment problem and represents it as an integer linear program that is approached with the help of Lagrangian relaxation. MI-GRAAL [66] uses seed-and-extend strategy together with the Hungarian algorithm to construct weighted bipartite graphs. An alignment is built as a result of solving the maximum-weight matching on these graphs. C-GRAAL [5] also exploits seed-and-extend strategy and alliteratively aligns common neighbors of already aligned nodes. Along with introducing the spectral signatures to measure topological similarity between subgraphs, GHOST [71] combines a seed-and-extend global alignment phase, where neighborhoods are matched by computing an approximate solution to the quadratic assignment problem, with a local search procedure. An even more sophisticated method, SPINAL [70], first builds a coarse-grained alignment and then improves it using seed-and-extend with local refinements. A very fast and efficient method called NETAL [78] uses a greedy approach to build an alignment from scoring matrices iteratively updating them. Table 7.1 provides short summary of the methods.

All approaches struggle to provide high-quality results on the huge, yet constantly growing, biological networks that we are confronted with nowadays. On real PPI networks, SPINAL as well as the GRAAL collection, proved to perform best and to offer biologically meaningful alignments. A brief comparison previously used in [66] is given in Table 7.2. Instead of replicating the conclusion

Tab. 7.1: An overview of the methods for the pairwise global network alignment problem. Here $n := \max\{|V(G_1)|, |V(G_2)|\}$, $m := \max\{|E(G_1)|, |E(G_2)|\}$, d is the largest degree of vertices in G_1 and G_2

Tool	Keywords	Running time
NABEECO	Bee Colony Optimization, Graph Edit Distance	$O(I \cdot P \cdot n \cdot d)$, here I is the number of iterations and P is the population size, $O(n^5)$ to compute graphlet degree signatures
GEDEVO	evolutionary algorithm, Graph Edit Distance	$O(I \cdot P \cdot n \cdot (\log n + d))$, here I is the number of iterations and P is the population size, $O(n^5)$ to compute graphlet degree signatures
IsoRank	eigenvalue problem on pairwise node similarity matrix	
GRAAL	seed-and-extend, greedy	$O(n^2 \log n + n \cdot m)$, $O(n^5)$ to compute graphlet degree signatures
H-GRAAL	maximum-weight matching in bipartite graph, Hungarian algorithm	$O(n^3)$, $O(n^5)$ to compute graphlet degree signatures
MI-GRAAL	seed-and-extend, maximum-weight matching in a bipartite graph, Hungarian algorithm	$O(n \cdot (m + n \cdot \log n))$, $O(n^5)$ to compute graphlet degree signatures
C-GRAAL	seed-and-extend, greedy	$O(n^2 + m)$, $O(n^5)$ to compute graphlet degree signatures
PISwap	maximum-weight matching in a bipartite graph, local search	$O(\max\{c^4, d^2\} \cdot B \cdot \|M^*\|^3)$, B is the largest similarity value for two sequences, M^* is a maximum weighted bipartite mapping ($\|M^*\| \leq n$), c is the number of most highly weighted neighbors of a vertex $c \leq d$.
GHOST	seed-and-extend, local search	$O(d^4)$ spectral signatures, pseudo-polynomial
SPINAL	greedy matching in a bipartite graph	$O(n^2 \cdot (d^2 + \log n)$
NATALIE 2.0	integer linear programming formulation, Lagrangian relaxation	
PATH	concave and convex relaxations over doubly stochastic matrices	$O(n^7)$
GA	a gradient ascent method on relaxation over doubly stochastic matrices	$O(I \cdot n^3)$, where I is the number of iterations
NETAL	greedy	$O(n^2 \cdot \log n + m^2 + n \cdot m \cdot \log n)$, or $O(n^2 \cdot \log^2 n)$ for PPI networks

Tab. 7.2: The highest edge correctness (EC) achieved by different tools aligning two pairs of networks, adopted from [66] and extended by the results of SPINAL and GHOST. Note that GHOST did not terminate for *yeast2* vs. *human1*. Note that GEDEVO obtained better results (refer to Table 9.1 in Chapter 9; Table 8.2 summarizes all data sets). The definition of EC and description of the networks can be found in the next chapter on page 110 and page 112, respectively.

	IsoRank [67]	GRAAL [68]	H-GRAAL [69]	MI-GRAAL [66]	C-GRAAL [5]	SPINAL [70]	GHOST [71]
yeast2 vs. *human1*	3.89	11.72	10.92	23.26	22.55	19.33	-
Meso vs. *Syne*	5.33	11.25	4.59	41.79	26.02	25.86	41.98

from the above cited papers that Network Alignment offers biological insights, we focus on the methodological problem that the existing tools possess. As we will demonstrate in Chapter 9, many of the existing software tools cannot cope well with such big networks. This becomes most evident when we see them fail on aligning a network to itself, which should result in 100% of aligned edges. Regarding the generalization to the other types of networks, many of the proposed approaches cannot be directly applied to graphs other than undirected and unweighted.

7.3 Multiple Network Alignment

A natural generalization of the pairwise network alignment problem to multiple networks aims to find conserved subnetworks in several input networks. A set of techniques have recently been proposed for the multiple network alignment problem.

MULE [79] is one of the first heuristics for detecting frequently occurring interaction patterns and modules in biological networks. It is based on contraction orthologs that allows to substantially reduce sizes of input graphs simplifying them. NetworkBLAST-M [80] represents sets of orthologous proteins as single nodes in the network alignment graph and uses a seed-and-extend approach to build an alignment. A general framework implemented in C3Part-M [81] (see also [82]) avoids explicit construction of the multigraph that is built from the input networks. C3Part-M is an exact method based on extracting connected components from the multigraph. IsoRankN [83] is an extension of IsoRank [67] and is based on spectral clustering on the induced graph of pairwise alignment scores. Graemlin 2.0 [84] is a three-stage heuristic that requires information about phylogenetic relationship to learn parameters for its scoring function. SMETANA [85] aligns networks in two stages. It first computes node correspondence scores using a semi-Markov random walk model. Then, using these scores that serve as a probabilistic similarity measure between nodes, SMETANA finds the maximum expected accuracy alignments in a greedy manner. A method proposed in [86] first clusters proteins into groups based on their similarity and then aligns them in a seed-and-extend manner. Defining seeds as sets of proteins with high similarity scores, the method expands the seeds to conserve edges. Another similar heuristic, called BEAMS, was described in [87]. A nature inspired heuristic called NetCoffee [88] optimizes its target function using simulated annealing. The search space there is built with the help of triplets of weighted bipartite graphs.

All approaches, in contrast to the one that we will introduce in Chapter 10, define the multiple network alignment problem as a set of many-to-many mappings between the proteins in the different graphs, i.e. several proteins from one graph are allowed to be mapped to several proteins from other graphs. The intention behind this is to account for gene duplication events and to maximize biological significance scores, which are usually measured by Gene Ontology agreement of the grouped nodes (see for example [89]). A recent review of existing tools in the field of network alignment may be found in [90] and [91].

Note a problem of finding a so-called prototype graph from the pattern recognition field is similar to the multiple network alignment problem (see for example [92]). There, a prototype graph is usually defined as a graph that minimizes the distance to all elements it represents and is used in clustering or classification problems. However, many of the methods developed for this problem were tested on quite small graphs often with up to 10 vertices only.

8 Network Alignment with Bee Colony Optimization Strategy

In this chapter we first formally define Network Alignment and then present NABEECO, a novel method for PPI network alignment, which is the first algorithm that is based on bee colony strategy minimizing the graph edit distance as optimization criterion. In the Section 8.2 we describe the data that is used to evaluate our tools and that consists of real PPI networks from different species, amongst them the same that were used in [66] to demonstrate MI-GRAAL's performance. Then, we describe the strategy behind NABEECO and evaluate it against MI-GRAAL. Our implementation of NABEECO as well as all used data sets are publicly available at `http://nabeeco.mpi-inf.mpg.de`.

8.1 Problem definition

We consider PPI networks as undirected unweighted graphs. An *graph* G is a pair $(V(G), E(G))$ with $E(G) \subseteq V \times V$; the elements of $V(G)$ are called vertices of the graph G, the elements of $E(G)$ are edges. The *size* of a graph G is the number of edges in $E(G)$, denoted by $|E(G)|$. The neighborhood of a vertex v in a graph G, denoted by $N_G(v)$, is the set of all vertices adjacent to v. The *degree* of v is $|N_G(v)|$.

Consider two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and a one-to-one mapping $f : V_1 \to V_2$. By GED$'$ we denote the *graph edit distance between* G_1 and G_2 *induced* by mapping f that is computed as follows:

$$\begin{aligned}\mathrm{GED}'_f(G_1, G_2) := |\{(u, v) \in E_1 : (f(u), f(v)) \notin E_2\} \cup \\ \{(u', v') \in E_2 : (f^{-1}(u'), f^{-1}(v')) \notin E_1\}| \ .\end{aligned}$$

By definition, $\mathrm{GED}'_f(G_1, G_2)$ counts inserted or deleted edges induced by the mapping f to make G_1 isomorphic to G_2, and it can easily be extended to reflect vertex/edge dissimilarities or any other related information (e.g. protein sequence similarity) of the underlying networks.

In *Network Alignment*, we aim to find a mapping f that minimizes the $\mathrm{GED}'_f(G_1, G_2)$.

In previous work, the quality of the mapping f of most biological network aligners is assessed by using the number of aligned (shared) interactions, defined as:

$$|\{(u, v) \in E_1 : (f(u), f(v)) \in E_2\}|$$

or the number of conserved interactions, defined as:

$$|\{(u, v) \in E_1 : \mathrm{dist}(v, f(v)) < \Delta, \mathrm{dist}(u, f(u)) < \Delta, (f(u), f(v)) \in E_2\}| \ ,$$

where $\mathrm{dist}(x, y)$ is a dissimilarity between $x \in V_1$ and $y \in V_2$ (such as BLAST E-value), and Δ is the node dissimilarity threshold. This corresponds to the intuition that the closer two species in the evolutionary tree are, the higher the number of conserved interaction partners they share. Also note, while very high protein sequence similarities, may often serve as a strong indication for the correspondence of the proteins, the relation between proteins with lower sequence similarities is difficult to recover [10]. Though comparison of 3D structures of the proteins is a much better way to discover such relations, only for relatively few proteins their 3D structures were determined. At the same time, incorporating reliable external biological information can relax the Network Alignment problem by substantially reducing the search space by, for example, predefining preferable sets of nodes to be mapped. Although NABEECO as well as GEDEVO can include such external information, a “good” method should be able to determine an optimal mapping, by maximizing the number of shared interactions and thus utilizing the graph structure alone. For this reason and to assure comparability between the existing biological Network Alignment tools we focus on topological criteria only.

Aligning using topological information, the performance of many methods is assessed with the so-called *Edge Correctness* (*EC*), which is particularly useful when comparing many pairs of networks with different sizes. Edge Correctness $\mathrm{EC}_f(G_1, G_2)$ corresponds to the proportion of aligned edges and is defined as follows:

$$\mathrm{EC}_f(G_1, G_2) := \frac{|\{(v, u) \in E_1 : (f(v), f(u)) \in E_2\}|}{\min(|E_1|, |E_2|)} \cdot 100[\%] \ ,$$

The highest value of EC is 100% and occurs if one input graph is a subgraph of the other graph. Other indicators of topological quality on an alignment, such

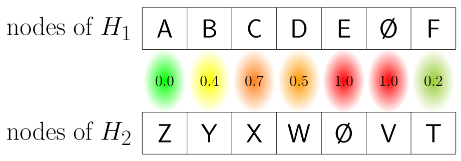


Fig. 8.1: A mapping between graphs H_1 (vertices A, B, C, D, E, and F) and H_2 (vertices T, V, W, X, Y, and Z) with arbitrary pair scores (for illustration only). In this mapping, A fits perfectly to Z, C corresponds quite poorly to X, E is deleted and V is inserted and both have worst pair scores. One major principle behind the generation of new individuals in NABEECO and GEDEVO is to swap pairs of vertices with bad pair scores.

as size of the largest connected component of a graph build on aligned edges or induced conserved structure score were also considered.

Note that the methods we describe in this thesis (NABEECO, GEDEVO, and GEDEVO-M) internally utilizes the graph edit distance as optimization criterion, not the Edge Correctness. This makes them more applicable to general graph comparison problems outside computational biology. However, if we set the costs for vertex deletions/insertions/substitutions and edge substitution to zero but only the cost for edge deletions/insertions to one, the EC is be related to GED as:

$$\mathrm{EC}_f(G_1, G_2) = \frac{|E_1| + |E_2| - \mathrm{GED}'_f(G_1, G_2)}{2 \cdot \min(|E_1|, |E_2|)} \cdot 100[\%] \ .$$

This allows us to compare GEDEVO to existing approaches on protein-protein interaction Network Alignment based on the EC criterion, as in previous works, which is particularly useful for graphs where differences between sizes are common (as in PPI networks from different organisms).

Based on a set of alignments (one-to-one mappings), the core idea of our methods is to generate a set of new alignments so that the chance of obtaining better mappings is greater than by random choice. To guide the search process of our methods we use pair scores (see Fig. 8.1). For a vertex $v \in V(G_1)$, the *pair score* is defined as:

$$p_f(v) := \frac{\begin{array}{c}|\{u \in N_{G_1}(v) : f(u) \notin N_{G_2}(f(v))\} \cup \\ \{u' \in N_{G_2}(f(v)) : f^{-1}(u') \notin N_{G_1}(v)\}|\end{array}}{|N_{G_1}(v)| + |N_{G_2}(f(v))|} + d(v, f(v)) \ ,$$

where $d(v, v')$ measures the difference between $v \in G_1$ and $v' \in G_2$. By the definition, the pair score $p_f(v)$ is the sum of an external distance between the two vertices and the relative number of edge deletions/insertions induced by the mapping f. Intuitively, this "pair score" reflects the contribution of the vertices

Tab. 8.1: Summary of PPI networks used for evaluations.

Short name	Species	Citation	Proteins	Interactions
cjejuni	*Campylobacter jejuni*	[96]	1095	2988
Meso	*Mesorhizobium loti*	[97]	1803	3094
Syne	*Synechocystis sp.(PCC6803)*	[98]	1908	3102
ecoli_fi	*Escherichia coli*	[99]	1941	3989
yeast2	*Saccharomyces cerevisiae*	[100]	2390	16 127
SC	*Saccharomyces cerevisiae*	[101]	5152	24 847
HS	*Homo Sapiens*	[101]	5878	14 015
DM	*Drosophila Melanogaster*	[101]	7533	22 477
ulitsky	*Homo Sapiens*	[102]	7384	23 462
human1	*Homo Sapiens*	[103]	9141	41 456
hprd	*Homo Sapiens*	[104]	9672	37 047

to the GED′ of the mapping and thus its quality: The lower the pair score, the better vertex v fits vertex $f(v)$.

The first term of a pair score penalizes the unmatched edges in both networks. As a measure between vertices (the second term of the pair score) of the input graphs we use graphlet degree signature distance (GSD) (see [93] and [94]). It restricts our search method to the graph topology and is thus independent from any external knowledge (protein similarities) other than the two input graphs. GSD can be considered as the magnitude of difference in local topology of two nodes from different graphs. Computing all graphlet degree vectors of a graph, needed for GSD, requires $O(n^5)$ time, where n is number of vertices in the graph. Despite the high theoretical running time, the vectors for the special case of PPI networks (which are relatively sparse) can be obtained in reasonable time, especially if compared to the run time for approximating solution of the Graph Edit Distance problem. This, however, does not necessary hold for more dense graphs. Our tools combine the precomputed GSDs (local: the difference between two vertices) with the GED′ (global: the number of vertex/edge deletion/insertions induced by a mapping). To compute graphlet degree vectors and the corresponding distances we used our own software implemented in C/C++.

Note other measures and their combination can also be used to compute pair scores, for example, spectral signatures [71], BLAST E-values [9], protein secondary/3D structure similarities, topological similarities [95].

8.2 Data

For the evaluation of NABEECO as well as GEDEVO in Chapter 9 with existing tools we used several PPI networks (see Table 8.1). The following six networks were previously used for evaluating C-GRAAL and MI-GRAAL. The two bacterial networks *cjejuni* and *ecoli_fi* are well-studied high-confidence networks: The first network resulted from high-throughput yeast two-hybrid screens; the second network was constructed using experimental and computational data (see [99]). The *Syne* network was obtained through a modified high-throughput yeast two-hybrid assay and covers around half (52%) of the total protein coding genes; similarly for network *Meso* that involves 24% of the protein coding genes. The high-confidence network *human1* was created by combining data from multiple sources including HPRD [104]. The network from [100] is based on (post-processed) data from high throughput experiments.

In addition, we obtained the networks *DM*, *SC*, and *HS* from the DIP database, which contains experimentally determined and manually curated protein interactions. The *hprd* network is a PPI network obtained from the Human Protein Reference Database (HPRD), which is a repository storing high-quality manually curated human interaction data. The human interactome network *ulitsky* is a compilation of protein-protein interactions, based mostly on small-scale experiments, from several interaction databases, including the HPRD database. Refer to Table 8.1 for a summary and citations.

8.3 Methods

Bee Colony Optimization (BCO) is a population based nature-inspired heuristic for solving hard optimization problems [105, 106]. BCO mimics behavior of a real honey bee colony in a hive; the general procedure of BCO is given in Algorithm 1.

In abstraction from a real honey bee colony, the artificial concept allows for 'only' two types of bees: scouts and worker bees. Every bee corresponds to a point (food source) in the solution space (set of fields with food sources) of the problem. Scouts are to explore a new horizon, keeping updated the pool of food sources known to the bees in the hive. The behavior of worker bees, which are gathering the food, corresponds to local exploration of the available known fields. A food source points to a solution, which is, in the context of Network Alignment, a mapping between two graphs. In the hive all bees share the information about the properties of the sources they know about. This information influences the choice of worker bees when they decide which field to fly to in the next round: The higher the quality of the food source, the higher

```
generate initial population
repeat
    send scout and workers
    share information
    gather new solutions
    memorize the best solution achieved
until termination
```

Algorithm 1: General Scheme of a Bee Colony Optimization Algorithm

its chances to be picked up as a target. Having chosen a target, a worker bee does not fly to the very point of the solution but to a location close to it. The procedure is repeated multiple times.

The quality of a food source is composed of two factors: the quality of the solution, associated with the food source, and the amount of food at the specific location, which decreases over time when it is picked as a target. The former is the actual criterion we aim to optimize, the graph edit distance.

8.3.1 Initialization Step

The better the initialization the higher the chance for having a 'good' starting point for the later steps in the optimization process. Along with uniform random mappings, NABEECO uses GSD to greedily create a set of mappings in which vertex pairs have similar local topologies. Use of more elaborated heuristic may provide even better initial population (in cost of running time) and may be included in the future.

8.3.2 Solution Gathering Step

The 'gathering' step performed by worker bees is the most crucial step in BCO, and has to guarantee a certain degree of diversity regarding the traversed solutions but, at the same time, being restrictive enough. In contrast to scouts, which generate mappings randomly and keep updating the solution pool, in NABEECO workers rely on multiple techniques to explore the local neighborhood of the solutions they have chosen as targets:

- *Greedy swaps* randomly pick k_1 of m vertices with the worst pair scores and exchanges them pairwise in greedy manner to improve the total GED of the mapping. Here, we set $k_1 := 6$ and $m := 20$.
- *Local permutation* of k_2 vertices performs all $k_2!$ permutations and accepts the mapping with the best GED. We set $k_2 := 6$ here. Two thirds of these k_2 vertices are chosen from the m vertices with the worst pair scores, while the rest vertices are picked up randomly.

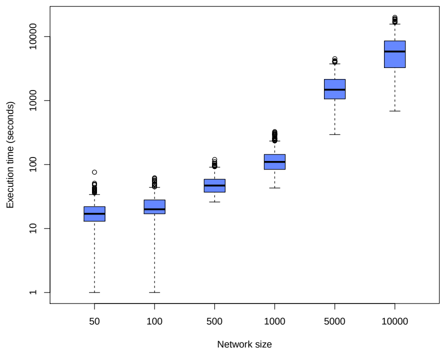


Fig. 8.2: Running time of NABEECO on artificially generated pairs of graphs. The x-axis denotes the number of vertices in the bigger graph.

- *Random swaps* randomly exchange $k_3 := 5$ pairs of vertices irrespective of the resulting GED.
- *Random greedy swaps* are executed similarly to greedy swaps, but applied among all vertices of a given mapping.

Each round one of the methods is chosen with a 33% chance, except for the relatively expensive *local permutation*, which is executed with a probability of 1%. These local exploration operations are implemented to avoid local optima but speed-up convergence.

8.3.3 Termination Step

For NABEECO, similar to many other bio-inspired heuristics, the exact running time for finding exact solutions is difficult to estimate. Convergence mainly depends on the input as well as the population size. The program can be executed for a predefined number of iterations or a preset run time. Another option is to set a no-change-in-quality threshold: If the GED does not improve for a certain number of iterations (convergence time), the program is considered to be converged and stopped.

The running time of NABEECO depends on the time needed to perform operators on the input graphs as well as the number of iterations and the population size. Computing values for pairs scores of a mapping requires $O(n \cdot d)$, where n is the number of vertices in the input graphs and d is the highest degree. In greedy swaps and random greedy swaps, to find m vertices with the worst values of pair scores in the mapping requires $O(m \cdot n)$ time. In local permutations and random swaps there are $k_2!$ and k_3 evaluations of the mappings,

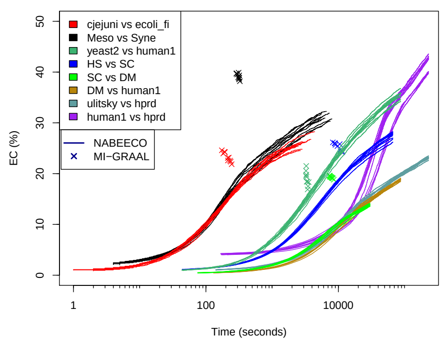


Fig. 8.3: Quality (Edge Correctness) vs. run time for aligning protein-protein interaction networks. Results achieved by MI-GRAAL are shown with crosses. Each line/cross represents one run.

respectively. In every iteration of NABEECO at least one evaluation of a solution can be required. Thus, given that NABEECO is set to run I iterations with the population of size P and the parameters k_1, k_2, m are bounded by a constant, the total running time of NABEECO is $O(I \cdot P \cdot n \cdot d)$.

To evaluate convergence time for varying input sizes we generated a set of random 'PPI-like' graphs of different sizes by using the preferential attachment mechanism [107]. In addition, we modified the graphs by inserting and deleting a predefined number of edges randomly. The run times for pairs of graphs with different number of vertices size are depicted in Fig. 8.2. Termination criterion (convergence time) was set to stop after 1000 iteration with no improvements of the best GED′ and the population size was set to 500 bees.

8.4 Evaluation

In Fig. 8.3 we summarize the results for MI-GRAAL and NABEECO on comparing pairs of the PPI networks from Table 8.1. The highest EC value for all pairs of graphs are further summarized in Table 8.2.

Since both NABEECO and MI-GRAAL have elements of randomness they were run 10 times for each input pair of graphs. All runs were executed on a 64 bit Linux 2.6.32 kernel, running on an Intel Xeon CPU W3550 @ 3.07GHz and 12 GB RAM. Time needed to compute graphlet signature vectors is not taken into account both for NABEECO and MI-GRAAL. The population size in NABEECO was set to 500 bees, with 10% of them as scouts. Since MI-GRAAL does not provide intermediate results, the final EC for each run is depicted as one cross. For one NABEECO run, we can plot the quality of intermediate

Tab. 8.2: The highest quality (Edge Correctness) values from pairwise alignments of different protein-protein interaction networks with NABEECO and MI-GRAAL.

		EC (%)	
Network 1	Network 2	NABEECO	MI-GRAAL
ecoli_fi	*cjejuni*	**28.24**	24.60
Meso	*Syne*	32.25	**39.88**
yeast2	*human1*	**36.78**	21.38
HS	*SC*	**28.26**	26.15
SC	*DM*	14.14	**17.73**
DM	*human1*	**19.09**	-
ulitsky	*hprd*	**23.51**	-
human1	*hprd*	**43.57**	-

results (as edge correctness of the best found solution so far). Observe that an EC is constantly improving over time.

Quality-wise both, MI-GRAAL as well as NABEECO, perform almost equally well. For comparing the smaller PPI networks from *cjejuni*, *Meso*, *Syne*, *ecoli_fi*, MI-GRAAL gives better results in shorter time than NABEECO. For comparing larger networks, however, NABEECO is faster and converges to results of better quality (higher EC). Note that MI-GRAAL failed aligning the human (bigger) graph pairs.

Despite competitive results, the main problem of NABEECO is that its operators modify relatively small portions of a solution mapping. This leads to too slow local improvements of the known solutions and getting into local minima. In the next chapter we describe another nature inspired heuristic, called GEDEVO, that allowed us to diversify the pool of solutions and obtain even better alignments on these networks. We also compare GEDEVO with other recent tools for network alignment.

9 Evolutionary Algorithm for Network Alignment

In this chapter we describe GEDEVO, a novel method for PPI Network Alignment. GEDEVO is an evolutionary algorithm that uses the Graph Edit Distance problem as optimization model for finding the best alignments. For evaluation, we use a set of high quality PPI networks (see Section 8.2 in Chapter 8), including the same networks previously used for C-GRAAL [5] and MI-GRAAL [66] for comparison with existing tools. We will demonstrate that GEDEVO performs comparable or better than recent tools exploiting topological information only, being at the same time fast and flexible. An implementation of GEDEVO as well as all used data sets are publicly available under `http://gedevo.mpi-inf.mpg.de`.

9.1 Methods

Evolutionary Algorithms (EAs) are nature inspired heuristics, which are widely used to tackle many NP-hard problems (see for example [108, 109]). The key idea behind EAs is mimicking the rule "survival-of-the-fittest" on a population of different individuals. Note multiple EAs were suggested for a number of formulations of the Graph Matching problem (see for example [110, 111, 112]). However, the major hindrance for efficient EA-based combinatorial optimization remained unsolved: the generation of new individuals. In the following we briefly introduce a general scheme of an EA and describe how we modified it with partial adoptions from [110] to Network Alignment.

In an EA an individual represents a solution for the problem, i.e. a mapping between vertices of two graphs (see Figure 8.1 in Chapter 8). The state of an individual determines how well the individual fits to the requirements of the environment it populates. New individuals result from inheriting parts of solutions from its parents; the better an individual fits the requirements, the higher are its

```
generate initial population
repeat
    for multiple times do
        generate a new individual using an evolutionary operator
    end
    evaluate all individuals
    apply selection criterion
until termination
```

Algorithm 2: General scheme of an Evolutionary Algorithm

chances to survive and to pass its solution to future generations. Mutations of the solutions exposed to a new individual are another way of mimicking nature in EAs. The requirements of the environment are related to the fitness function, for which we utilize GED′ defined in Section 8.1. Starting with generating a (quasi) random initial population, an EA repeats the following three steps, until a termination criterion is met: offspring generation, individual evaluation, survival function application.

9.1.1 Initial Population Generation and Evaluation of an Individual

An individual represents a mapping f. Individuals in the initial population are created with random permutations. However, initialization in a more sophisticated manner, which, as a consequence, will require more time, may reduce the convergence time of the algorithm. Here, we may use protein sequence similarities, acquired by BLAST [9], for instance.

The score of an individual together with its *health*, a non-increasing function of the number of iterations and GED′ of the individual, defines its fitness. The introduction of health allows keeping individuals with a "bad" GED′ for a number of iterations instead of simply discarding them immediately. This introduces some divergence and contributes to avoiding local optima.

9.1.2 Offspring generation

To generate new individuals we combine a set of different operations to balance between a reasonably high population diversity to avoid local optima and a high and fast convergence towards optimal solutions. The operations are as follows:

- *Random generation* creates an individual by relating it to a mapping based on a random permutation; it requires $O(n)$ time.

- In *PMX-like mutation* we adopt the idea of partially-mapped crossover (PMX), initially introduced in [113]. We partition a mapping into two sets of pairs, low scores and high scores, by using the average over all pair scores in the mapping as a threshold. Afterwards, the high scoring pairs are swapped randomly. To avoid local minima, however, we also swap low scoring pairs with a low probability (of 1% for GEDEVO and PPI networks). PMX-like mutation evaluates each pair by using the pairScore, which requires at most $O(n \cdot d)$ time.
- A so-called *crossover* results in an individual that in the first place preserves pairs with low pair scores from two or more parents. Ties are resolved randomly. Crossover is similar to the previous operation with a term responsible for sorting n pairs from a constant number of parents $p \leq 8$, which results in $O(p \cdot (n \cdot d) + p \cdot n \cdot \log(p \cdot n)) = O(n \cdot (\log n + d))$ time.
- With *directed mutations* we swap of a number $r \leq 20$ randomly chosen "bad" pairs in the mapping of an individual. At the end, the one swap that induces the best score is kept. One swap requires recomputing two pair scores. Thus, the running time of the operation is bound by $O(r \cdot 2 \cdot n \cdot (d + s)) = O(n \cdot d)$.

These operations are GEDEVO's strategies to find and keep "good" pairs while a "bad" pair is swapped more often with another "bad" pair, in this way improving the final score of the mapping. Over a number of iterations, many individuals are exposed to these operations by GEDEVO to traverses the search space and optimizes the final score.

9.1.3 Termination and Running Time

No practical exact algorithm for computing optimal value of the graph edit distance problem on large graphs exists. Consequently, it is hard to theoretically estimate the number of necessary iterations until a "good" solution can be achieved. Convergence time mainly depends on the population size as well as on the input graphs' topological properties. Our implementation of GEDEVO can be set to execute (1) a specified number of iterations, (2) a pre-specified running time, or (3) a fixed number of iterations of no significant changes in the mapping scores of the best individuals (such that convergence was probably reached).

The total theoretical running time of GEDEVO is based on the run times of the individual steps. The evaluation step is performed in $O(P \cdot n \cdot d)$, with P is the population size. The offspring generation step requires $O(P \cdot (n + n \cdot d + n \cdot (\log n + d) + n \cdot d)) = O(P \cdot n \cdot (\log n + d))$ time. The selection step sorts the individuals from the older and new generations in $O(2 \cdot P \cdot \log(2 \cdot P))$

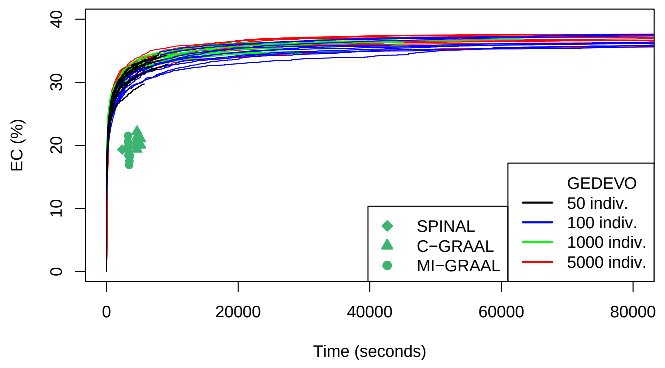


Fig. 9.1: The influence of the population size to the performance of GEDEVO aligning *yeast2* vs. *human1* in comparison to SPINAL, C-GRAAL and MI-GRAAL. Each line/symbol represents one run.

time. Given that GEDEVO runs I iterations, its total running time sums up to $O(I \cdot P \cdot n \cdot (\log n + d))$.

9.2 Evaluation

Here, we evaluate GEDEVO against the four tools GHOST, SPINAL, C-GRAAL and MI-GRAAL, which form the current state of the art and have been shown to outperform other existing tools [70, 66, 5].

All tools were executed on a 64 bit Linux 2.6.32 kernel, running on an Intel Xeon CPU W3550 @ 3.07GHz and 12 GB RAM. SPINAL is deterministic and was thus executed only one time for each pair of the input networks, while MI-GRAAL, GHOST, C-GRAAL and GEDEVO, as randomized algorithms, we executed 10 times for each pair. The execution of all tools was interrupted after 24 hours of runtime without termination. MI-GRAAL and C-GRAAL, similarly to GHOST, require graphlet degree signatures as preliminary node similarity measures, which were precomputed and used as input (precomputation time not taken into account for evaluation). The termination criterion for GEDEVO was set to stop after 3000 iterations of no significant improvement of GED′ amongst the best solutions (individuals).

In Fig. 9.1 we depict the influence of the population size to the progression of the EC (convergence). Runs with 50 individuals (black line) converged earlier to the final solution, still providing quite high values of EC. With larger population sizes the runs obtained slightly better alignments with higher EC and

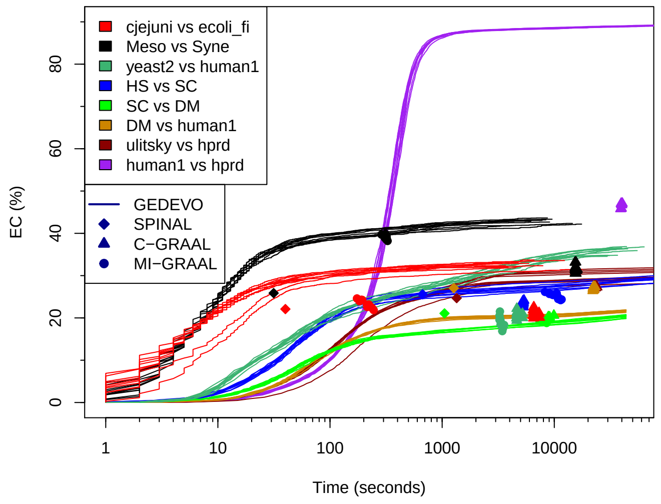


Fig. 9.2: Convergence and Edge Correctness vs. run time for aligning different PPI networks, 10 runs for each tool. Each line/symbol represents one run.

reached them slightly faster. This indicates that GEDEVO is quite robust to different population sizes, given that they are reasonably large. In the remaining evaluations (that are described below) we used 500 individuals per run.

We executed GEDEVO and the four competing tools on multiple pairs of networks from Table 8.2 in Chapter 8. The resulting edge correctnesses and the according run times for all tools are depicted in Fig. 9.2. Since SPINAL, C-GRAAL and MI-GRAAL do not provide intermediate results, the final values are shown point-wise (diamonds, triangles, and circles); for GEDEVO the progression of EC is depicted with lines. The plot illustrates that GEDEVO can provide a “good” solution comparably fast. A summary of the maximal EC values from the plot is given in Table 9.1. Unsuccessful runs (no termination after 24 hours) of SPINAL and MI-GRAAL are marked with an “x”). Note: Since GHOST only terminated for the alignment of the two small networks *Meso* and *Syne* (with best EC: 41.98, runtime: 140 sec) we did not add it to Table 8.2 and Fig. 9.2.

The networks *human1*, *hprd* and *ulitsky* are all human PPI networks, therefore the EC scores for GEDEVO are comparably high. The results for aligning *ulitsky* with *human1* or *hprd* are rather “poor” since *ulitsky* is a compilation of data from different databases. The known overlap of *ulitsky* with *hprd* is only 649 nodes and 15 305 interactions, from which GEDEVO aligned 11 800.

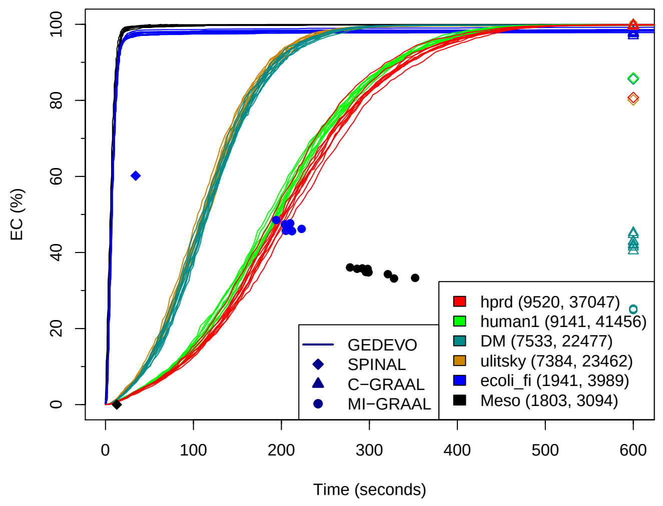


Fig. 9.3: Aligning a network against itself should result in an Edge Correctness of 100%, which is achieved with GEDEVO and C-GRAAL in most cases (see text). Each line/symbol represents one run. Unfilled symbols (at the right side of the plot) mean that it took more than 10 minutes to achieve the corresponding EC values.

To further investigate the robustness of the four methods on graphs where we definitely know the correct solution, we aligned some PPI networks against themselves. Naturally, this should result in an EC of 100%. In Fig. 9.2, we plot the EC vs. run time for the following data sets: *Meso*, *ecoli_fi*, *ulitsky*, *DM*, and *human1*. Note that GHOST only terminated for the self-alignment of the two smallest networks *Meso* (with best EC: 100%, runtime: 197 sec) and *ecoli_fi* (with best EC: 100%, runtime: 173 sec). We also downloaded and tested Natalie 2.0 [76] on our servers. It terminated with memory faults for all network pairs but the two smallest ones: For *cjejuni* vs. *ecoli_fi* (runtime: 7 hours) and self-alignment of *Meso* (runtime: 11 hours) the tool resulted with edge correctnesses of 97.64% and 20.38% respectively. Hence, we did not include GHOST and NATALIE 2.0 with Figure 4. Further note that a set of methods exists that restrict alignment candidates to a set of pre-mapped nodes (limited search space), see for example [77] and [114]. GEDEVO can be restricted to such pre-mappings (e.g. with BLAST as preprocessing) but it does not rely on it.

In conclusion, GEDEVO, in contrast to the other approaches, was able to achieve the expected 100% EC in all cases, often even faster than the existing

Tab. 9.1: The highest achieved Edge Correctness (EC) quality scores for alignments of different PPI networks from Fig. 9.2.

		EC (%)			
Network 1	Network 2	GEDEVO	MI-GRAAL	C-GRAAL	SPINAL
cjejuni	*ecoli_fi*	**33.70**	*24.60*	22.56	22.09
Meso	*Syne*	**43.60**	*39.88*	33.19	25.86
yeast2	*human1*	**38.14**	21.38	*22.20*	19.33
HS	*SC*	**30.40**	*26.15*	24.15	25.59
SC	*DM*	*20.79*	17.73	20.59	**21.07**
DM	*human1*	21.88	x	**27.36**	*27.04*
ulitsky	*hprd*	**32.00**	x	*27.56*	24.68
human1	*hprd*	**89.37**	x	*47.07*	x

tools. C-GRAAL reached around 97-98% of EC in most cases, but required up to 11 hours for the biggest networks (*hprd*, *human*), for which GEDEVO needed only approx. 10 minutes.

To sum up, in almost all cases, GEDEVO outperformed SPINAL, GHOST, C-GRAAL and MI-GRAAL in terms of quality and run time. Moreover, in contrast to the other methods GEDEVO was able to recognize the high similarity (*human1* vs. *hprd*) and composition (*ulitsky* vs. *hprd*) between the human PPI networks using topological information only. In addition, we wish to emphasize that GEDEVO provides intermediate results that allow for a manual termination of the software at earlier iterations when a high EC score (or a corresponding low GED′ value) has been found and convergence seems to be reached.

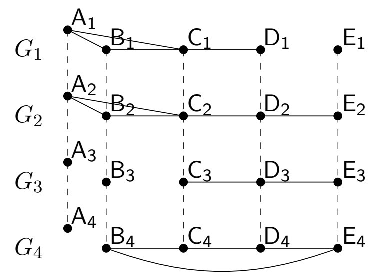
10 Evolutionary Algorithm for Multiple Network Alignment

In this chapter we describe GEDEVO-M a novel approach to the global multiple network alignment problem. In contrast to existing approaches, we do not want to allow for many-to-many mappings but require each node from each graph to be mapped to at most one node from each other graph. Our intention is to identify a solid 'core interactome', which we define as a set of nodes and interactions that we find topologically conserved in a set of multiple PPI networks. We extend the definition of the pairwise global network alignment defined in Chapter 8 and, in contrast to existing approaches, we seek to ensure that a network, if aligned to multiple instances of itself, will give a result where all nodes from all instances are aligned to its corresponding copies, i.e. in pairwise self-alignment. We will extend the graph edit distance model to multiple graph instances and introduce GEDEVO-M, an evolutionary algorithm that minimizes the corresponding optimization function. GEDEVO-M and all used data sets are publicly available at `http://gedevo.mpi-inf.mpg.de/multiple-network-alignment/`.

The rest of the chapter is organized as follows. We first give necessary notations and formally define the topological multiple network alignment problem. We also introduce several indicators that are useful for evaluating the resulting alignment. We further describe an evolutionary algorithm and its operators to approach the problem. Next, we present experimental results from aligning multiple real PPI networks with GEDEVO-M.

10.1 Problem definition

We adopt the notation introduced in Section 8.1 of Chapter 8. Let $\mathcal{G} = \{G_1, \dots, G_N\}$ be a set of N graphs. A *multiple mapping* (*m-mapping*) from G_1 to $\mathcal{G}' \subseteq$



$\text{GED}'_{f_{1,2}}(G_1,G_2)=1$, $\text{GED}'_{f_{1,3}}(G_1,G_3)=4$, $\text{GED}'_{f_{1,4}}(G_1,G_4)=4$, $\text{GED}'_{f_{2,3}}(G_2,G_3)=3$, $\text{GED}'_{f_{2,4}}(G_2,G_4)=3$, $\text{GED}'_{f_{3,4}}(G_3,G_4)=2$

$\text{mGED}'_{\mathcal{F}}=\sum_{f_{i,j}\in\mathcal{F}}\text{GED}'_{f_{i,j}}(G_i,G_j)=17$

4-AE $=1$, 4-AR $=\frac{1}{2}$,

3-AE $=2$, 3-AR $=\frac{3}{4}$,

2-AE $=2$, 2-AR $=\frac{5}{4}$

Fig. 10.1: An m-mapping $\mathcal{F}$ (dashed lines) on graphs G_1, G_2, G_3 and G_4. There is only one set of four edges that are mapped to each other in all graphs: the edges (C_1, D_1), (C_2, D_2), (C_3, D_3), (C_4, D_4), such that the frequency of edges aligning in all four graphs 4-AE equals to 1. The smallest number of edges in one of the input graphs is two, such that we have 4-AR=4-AE/2=1/2; similarly for the remaining values of k-AE and k-AR, for $k=\{2,3\}$.

$\mathcal{G}\setminus G_1$ is a set $\mathcal{F}'=\{f_{i,j}:G_i,G_j\in\mathcal{G}'\cup G_1\}$ of one-to-one mappings such that $f_{1,j}:V(G_1)\to V(G_j)$ and $f_{i,j}(v)=f_{1,j}(f_{1,i}^{-1}(v))$, for every $G_i,G_j\in\mathcal{G}'$ and $v\in G_i$. Given an m-mapping $\mathcal{F}'$, we define the *multiple graph edit distance* (*mGED*$'$) induced by $\mathcal{F}'$ as:

$$\text{mGED}'_{\mathcal{F}'}:=\sum_{f_{i,j}\in\mathcal{F}'}\text{GED}'_{f_{i,j}}(G_i,G_j)\ ,$$

and *multiple edge correctness* (*mEC*) is defined as:

$$\text{mEC}_{\mathcal{F}'}:=\frac{\sum_{f_{i,j}\in\mathcal{F}'}\text{EC}_{f_{i,j}}(G_i,G_j)}{|\mathcal{F}'|}\ .$$

We use the mGED$'$ to evaluate the quality of an m-mapping. However, mEC is more convenient to interpret, in particular on graphs of different sizes, especially given that the optimal mGED$'$ value (minimum) is unknown *a priori* while the mEC scales between 0 and 1. Given an m-mapping $\mathcal{F}$ on $\mathcal{G}$, a pair $v,u\in V(G_1)$, is *k-aligned* if $|\{(f_{1,j}(v),f_{1,j}(u))\in E(G_j):G_j\in\mathcal{G}\}|=k$. If $k>1$, we also refer to k-aligned pair as *k-aligned edge*. We denote the number of k-aligned pairs by k-AE. The *k-aligned edges ratio* is defined as:

$$k\text{-AR}:=\sum_{i=k}^{N}\frac{i\text{-AE}}{\max^i\{|E(G_1)|,\ldots,|E(G_N)|\}}\ ,$$

where $\max^i A$ is the i-th largest (or the $(N-i+1)$-th smallest) element in the set A. Given a *k-common graph* G that is built on the vertices that are incident to ℓ-aligned edges, for $\ell = k, \ldots, N$, the value of k-AR indicates the size of G opposed to the ideal case when $(k-1)$-many smallest graphs $\mathcal{G}'' \subset \mathcal{G}$ are all subgraphs of a graph $G \in \mathcal{G} \setminus \mathcal{G}''$. Given an m-mapping $\mathcal{F}'$ to $\mathcal{G}'$ and $v \in V(G_1)$, a *tuple* is a set $\{f(v),\ \text{where } f : v \to V(G), G \in \mathcal{G}'\}$. A *tuple score* is defined as:

$$P_{\mathcal{F}'}(v) := \sum_{f \in \mathcal{F}'} p_f(v) \ ,$$

and *full score* S is defined as:

$$S := \sum_{v \in V(G_1)} P_{\mathcal{F}}(v) \ .$$

Given a set of graphs $\mathcal{G}$, we define the Topological Multiple Network Alignment problem (TMNA) as the problem of finding an m-mapping $\mathcal{F}$ on $\mathcal{G}$ such that $\text{mGED}'_{\mathcal{F}}$ is minimal over all possible m-mappings on $\mathcal{G}$. Clearly, TNMA is NP-hard even for two graphs. The intuition behind the formulation of TNMA is as follows. Ideally, a biologically meaningful alignment of multiple PPI networks should map functionally related proteins. At the same time, for closely related species, the overall difference in the corresponding PPI networks should be small due to evolutionarily conservation of many interactions (edges in the graphs). This is thought to be achievable if many pairs of vertices induce smaller numbers of edge insertions and deletions. Similarly to the graph isomorphism problem, in TNMA, when aligning N copies of a graph G, an optimal m-mapping should result in $|E(G)|$-many N-aligned edges and have mGED'$= 0$. However, with the current PPI data being noisy and incomplete, the requirement of strict correspondence of all edges is unrealistic. We therefore chose to model the multiple PPI network alignment as TNMA. Clearly, the definition of TMNA can be generalized to integrate external data such as edge weights and external node-to-node similarities (derived, for instance, from pairwise protein sequence alignments).

10.2 Methods

To approximate Topological Multiple Network Alignments we propose a heuristic based on an evolutionary algorithm that extends our approach described in Chapter 9 developed for the alignment of two networks. Note that testing all possible mappings is already unfeasible for the pairwise network alignment problem. In TMNA, we also observe exponential growth of the search space with respect to the number of input graphs N. The number of all possible m-mappings here corresponds to $O((n!)^N)$, where n is the number of vertices in the input graphs.

Next we describe evolutionary operators that are specific for GEDEVO-M and essential for its performance. In GEDEVO-M, generating a new population we chose randomly a set $\mathcal{G}' \subseteq \mathcal{G} \setminus G_1$ and apply the following operators on $\mathcal{G}'$:

- In a *PMX-like mutation* a new individual is created by copying the m-mapping of a randomly selected individual from the previous generation. We randomly swap ("bad") tuples on $\mathcal{G}'$ that have tuple scores higher than a certain threshold. The threshold is defined as the average over all tuple scores on $\mathcal{G}'$.

- In a *random mutation* the tuples on $\mathcal{G}'$ with lower than the threshold values of their tuple scores are also given a certain chance to be swapped.

- A *crossover operator* constructs a new m-mapping from two or more "parent" individuals of the previous generation. We first compute tuple scores for every possible subset of $\mathcal{G}$ and sort them. Then, starting with larger subsets of $\mathcal{G}$, we repeatedly iterate over the corresponding tuple scores and assign some of these tuples to the new m-mapping until every subset is considered. Subsets having identical sizes are processed in random order. To maintain consistency (one-to-one requirement) in the new m-mapping, we set the tuple only if none of its element was already assigned in the previous steps. Finally, the unassigned images of the m-mapping are distributed randomly. We exploit three versions of the crossover operator. The first combines two individuals that have better values of fitness function achieved so far. The other two versions generate m-mapping from two and 3-8 parents: the parents here are selected randomly.

- A *directed mutation* selects a tuple t corresponding to $\mathcal{G}'$ and greedily looks for another tuple t' such that when t and t' are swapped the resulting m-mapping has the best improvement of the fitness function.

- A *greedy mutation* randomly chooses a set of tuples on $\mathcal{G}'$ of size at most 10 and changes the m-mapping by testing every possible permutation of the chosen subset. The permutation that results in the best improvement of the fitness function is assigned to the m-mapping at the end.

- A *random m-mapping* results in random permutations of the tuples of $\mathcal{G}'$.

We use the score S as an indicator of the quality of an m-mapping. To increase population diversity, we also exploit the notion of the health of an individual, which is maximal when an individual is born and drops over time (number of iterations). If the health reaches or falls below zero, the individual

Tab. 10.1: Protein-protein interaction data sets used for evaluating GEDEVO-M.

Short name	Species	Proteins	Interactions	Citation
Dmela	*Drosophila melanogaster*	7569	23 017	[101]
Scere	*Saccharomyces cerevisiae*	5011	22 503	[101]
Celeg	*Caenorhabditis elegans*	2686	4079	[101]
Hsapi	*Homo sapiens*	2332	3305	[101]
cjejuni_hc	*Campylobacter jejuni*	1095	2988	[96]
Meso	*Mesorhizobium loti*	1803	3094	[97]
Syne	*Synechocystis sp.(PCC6803)*	1908	3102	[98]
ecoli_fi	*Escherichia coli*	1941	3989	[99]
hprd	*Homo sapiens*	9671	39 214	[104]

is discarded. The health of the best 10% (according to S) of individuals remains unmodified; the health of the remaining 90% of individuals are subject to the following linear reduction: The best individual there remains unchanged and the worst one loses 90% of maximal health. Consequently, individuals with higher (worse) values of S and poor health have lower chances to survive and contribute to the future generations.

Similarly to GEDEVO we utilize Graphlet-degree Signatures Distances (GDS). In pair scores and thus in the full score S we combine the accumulated GSDs of the m-mapping and its actual mGED$'$. The GSD serves as a topological indicator for the differences of the local neighborhoods of the potentially matched vertices. With the above defined set of operators and the score S describing the quality of an m-mapping, our evolutionary algorithm GEDEVO-M improves the resulting m-mappings by preferably keeping those elements of m-mappings unchanged that lead to the lower values of the fitness function and modify elements of the m-mappings that have higher ones.

The crossover operator is the most computationally expensive operation (more expensive than evaluation of an individual, random mapping and other operators). For an individual there are $T := n \cdot 2^{(N-1)}$ distinct tuples obtained from $2^{(N-1)}$ subsets of $\mathcal{G} \setminus G_1$ and the number n of vertices in input graphs. Computing a pair score requires $O(d)$ time, where d is the highest degree of vertices. Thus, computing tuple scores for π parents in the crossover operator requires $O(\pi \cdot T \cdot d)$ time. Sorting all the tuples can be performed in $O(\pi \cdot T \cdot \log(\pi \cdot T))$ time. Assigning tuples to the new mapping requires at most $2^{(N-1)}$ passes on $\pi \cdot T$ values of tuples scores. In the final step, filling unassigned vertices is performed in $O(n \cdot 2^{(N-1)})$ time. Thus, the overall running time of the crossover operator on $\pi \leq 8$ parents is $O(\pi \cdot T \cdot (d + \log(\pi \cdot T)))$, which is not larger than $O(n \cdot 2^N \cdot (d + N + \log n))$. Given, that the evolutionary algorithm runs I iterations with population size P, its overall running time corresponds to $O(I \cdot P \cdot n \cdot 2^N \cdot (d + N + \log n))$. As for memory consumption, each individual

Tab. 10.2: The values of mGED′, mEC and aligned pairs aligning two sets of bacterial and eukaryotic PPI networks. See text and Fig. 10.1 for an illustration of mGED′, mEC, 4-AE, 3-AE, and 2-AE.

	achieved		random		k-aligned edges		
Set of networks	mGED′	mEC	mGED′	mEC	4-AE	3-AE	2-AE
Dmela, *Scere*, *Celeg*, *Hsapi*	139 024	24.16%	155 796	0.18%	310	638	4657
cjejuni_hc, *Meso*, *Syno*, *ecoli_fi*	30 287	25.27%	39 327	0.53%	328	503	1139

Tab. 10.3: Proportions of aligned edge aligning two sets of bacterial and eukaryotic PPI networks. See text and Fig. 10.1 for an illustration of 4-AR, 3-AR, and 2-AR.

Set of networks	aligned edges	proportion of aligned edges	4-AR	3-AR	2-AR
Dmela, *Scere*, *Celeg*, *Hsapi*	12 468	23.57%	9.38%	23.24%	24.91%
cjejuni_hc, *Meso*, *Syno*, *ecoli_fi*	5099	38.71%	10.98%	26.86%	63.51%

requires $O(n)$ space to represent every mapping from G_1 to $G' \in \mathcal{G} \setminus G_1$. A significant amount of memory is needed to store the values of the distance matrices (used in the pair score) between every pair of input graphs. Thus, the overall amount of required space results in $O(P \cdot N \cdot n + \binom{N}{2} \cdot n^2)$.

10.3 Data

To evaluate GEDEVO-M we used the same source of network data previously described in Chapter 8 (see Table 10.1 for the summary). The PPI networks of four eukaryotic species were extracted from the DIP database (release July 7th, 2013), which contains experimentally detected and manually curated protein-protein interactions [101].

10.4 Evaluation

In this section we provide experimental evaluation of the proposed algorithm. All executions of GEDEVO-M were performed on a 64-bit Linux 3.2.42 kernel running on AMD Opteron 6276 with 150GB RAM and 50 threads. From our testing experiments, despite the probabilistic nature, GEDEVO-M converged to similar mGEDs in all cases. We therefore present only the results for one GEDEVO-M run per input data set. The initial population size was set to 500

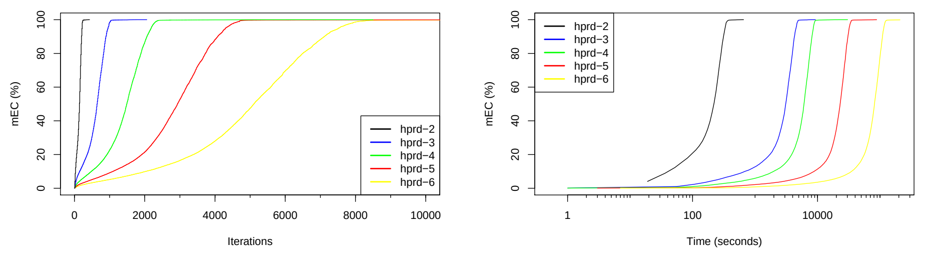


Fig. 10.2: Development of the multiple Edge Correctness (mEC) over progressing iterations (and elapsing time) while aligning multiple copies of the *hprd* network. Note that a perfect alignment of a network to multiple copies of itself should reach an edge correctness of 100%, which is reached by GEDEVO-M.

individuals. As the termination criterion we set GEDEVO-M to stop after 100k iterations or if no changes in the best value of mGED′ has been observed in in last 3k iterations.

We aligned two sets of PPI networks: “eukaryotic” (*Dmela*, *Scere*, *Celeg*, *Hsapi*) and “bacterial” (*cjejuni_hc*, *Meso*, *Syno*, *ecoli_fi*). The properties of the resulting alignments are summarized in Table 10.2. Note, the values of mEC for randomly generated m-mappings (which are also used for the initialization step) are just 0.18% and 0.53% for eukaryotic and bacterial networks, respectively. Thus, GEDEVO-M substantially improved these values to 25.16% and 25.27%, respectively. In Table 10.3, we provide edge alignment ratios and the proportions of aligned edges. Note that almost every fourth edge of the eukaryotic networks has an aligned edge; for the bacterial networks more than one third of edges is matched (38.71%).

The common graphs that are built on 4-, 3-, 2-aligned edges of the m-mapping on eukaryotic networks consist of 490, 1290, 4923 nodes, respectively. For the bacterial networks these values correspond to 511, 1057, 1565, respectively. The edges and vertices of the common graphs may give hints to the “core interactome”.

By the definition of TMNA, aligning multiple copies of a network the resulting mGED′ should be 0 with an mEC of 100%. Note that many tools for pairwise network alignment were not able to cope with the self-alignment. To demonstrate that GEDEVO-M is able to cope with this task on multiple networks, we aligned several copies of the comparably big *hprd* network. In Fig. 10.2, we show that GEDEVO-M successfully aligns multiple copies (i.e. 2-6x) of the *hprd* input network. In all tests GEDEVO-M achieved optimal values for mGED′

(0 edits) and mEC (100%). However, the time needed to reach this optimum grows exponentially with the number of input networks. This can be explained by the increasing number of the subsets of networks that can be generated when applying an evolutionary operator.

11 Discussions and Outlook

We presented several heuristics for Network Alignment and evaluated topological quality of the resulting alignments.

Our methods possess a set of advantages over the existing approaches:

- accuracy: NABEECO and GEDEVO perform similarly well but converges towards better solutions in shorter time on larger graphs.
- simplicity: The intuition behind the GED model is that for more similar networks there is a mapping that induces lower numbers of edge insertions and deletions. Correspondingly, a "good" mapping is composed of pairs of nodes that fit each other. To measure how good two nodes fit each other given a mapping we use the definition of the pair score, which in turn compounds the final criterion. The algorithms generate and improve mappings by applying a set of simple operators. The operators exploit the information about computed pair scores, and try to keep node pairs with better pair scores matched and get rid of those node pairs that do not fit each other well. As a whole, with the help of the defined operators on pairs and mappings, the algorithms are guided by these definitions and explore the search space correspondingly;
- scalability: For large-scale graphs our heuristics perform similarly well or better than existing tools; all the methods can already run on multiple cores and can naturally be parallelized to be able to run on multiple computational nodes that will allow to scale them to even larger graphs;
- flexibility: In contrast to many other methods and tools, NABEECO, GEDEVO and GEDEVO-M can work on topology only but they may also naturally incorporate non-topological data and can further be executed on any kind of graphs – labeled and directed, thereby generally allowing to compare, for instance, gene regulatory networks as well;
- diversity: While the most algorithms and tools for the network alignment problem result in a single solution mapping only, our approaches can produce multiple ranked candidates per a single execution run. This property

enables generation of more elaborated hypotheses about the true solution by analysis and comparison of the candidates.

While the overall running time and memory consumption on powerful computers are satisfactory, they can be one of the major hindrances for desktop computer users, in particular for bigger graphs: The higher hardware requirements are predictable due to the randomness of the methods: their performance depends on the population size and the number of iterations.

Next we outline further work required to improve applicability of our tools. First, there should be a possibility to incorporate "pre-matching" that is the knowledge of nodes that are known to correspond can be specified in the input. This feature will restrict the search space, and, more importantly, keep the global structure of the resulting alignment unchanged (compare to [115]). In our work we demonstrated the performance of our tools using topological information only. But it is also interesting to find proper transformations of the relevant biological data that can be incorporated into our tools in order to generate biologically more reliable alignments. The main difference between NABEECO and GEDEVO is that the former mainly focuses on local space exploration around found mappings, while the latter diversifies the know solutions to greater extent without extensive exploration of neighboring mappings. Therefore, it is interesting to see to what extent the performance of the current tools can be improved by combining their strategies and operators (similar to [116]).

Currently, the fitness function focuses on the number of edge insertions and deletions induced by the mapping and the operators designed to minimize this value. Depending on the amount of noise and completeness of the network this may result in a close to optimal mapping that does not preserve connectedness of the common subgraph. Therefore, a new criterion and operators should be designed that keep balance between GED$'$ and the connectedness of the solution mapping. Here, new operators should modify a mapping so that the nodes and their neighbors remain close to each other when mapped to the other graph. The closeness criteria can, for example, be derived from the definition of Neighborhood-Preserving Mapping (NPM) and Compactness-Preserving Mapping (CPM).

Moreover, a generalization of one-to-one node mappings to constrained many-to-many mappings is needed in order to address such evolutionary events as gene duplication and gene fusion, and to deliver better alignments. A many-to-many mapping here can, for example, be defined as a mapping that maps every node of the first network to at most two (or three) nodes of the second network and vice versa.

Despite the availability of developed methods and tools for the network alignment problem, there is a great amount of space for future studies. First, the research community will benefit from a platform that will provide a systematic and reproducible evaluation of the existing and future tools (similar to ClusEval [117]). The input data for the platform consists of standard real networks of various sizes modified by different models of noise. Aligning the input networks, the platform should be able to evaluate the influence of parameters on the topological and biological quality indicators of alignments. The topological quality indicators should include not only simple and important ones, such as Edge Correctness, but also more sophisticated indicators that, for example, show if the alignment keeps the overall network structure undestroyed (see the definitions of NPM and CPM in chapters 4- 5). For network pairs annotated with Gene Ontology (GO) terms the platform should not only count the numbers of overlapping GO terms to evaluate biological quality of alignments, but also use semantically more consistent measures, such as ones described in [89].

Secondly and more important, since it is rather impossible to reconstruct the actual process of evolution, the manual curation of resulting mappings is obligatory to create reliable and reusable alignments. Indeed, the ultimate aim of the network alignment problem is not to generate an alignment that optimizes a certain topological and biological criterion, but to find the actual correspondence between subnetworks of two species. Such correspondence enables understanding the generalities and differences between species and allows to reuse knowledge about one species for the other species in a reliable and accurate way. As for many bioinformatic problems, it is very unlikely that a fully automated and trustful tool for revealing ground truth in the network alignment problem will be developed in the nearest future. Therefore, the whole field and research community will benefit from the availability of a user friendly, flexible and customizable software tool that, rather than generate one single solution, will automate the process of assigning the correspondences between nodes of networks. Such a tool should not only visualize alignments, for example, as a plugin for Cytoscape [118], but also provide a possibility to incorporate different models and data to generate candidate mappings and to facilitate the process of reliable alignments in incremental and interactive manner. Note this prospective advantageously distinguishes our fairly general methods since at any moment of executions they provide a pool of different solutions that can be evaluated by a variety of models, helping to construct reliable alignments.

Bibliography

[1] Sayers, E.W., Barrett, T., Benson, D.A., Bolton, E., Bryant, S.H., Canese, K., Chetvernin, V., Church, D.M., DiCuccio, M., Federhen, S., Feolo, M., Fingerman, I.M., Geer, L.Y., Helmberg, W., Kapustin, Y., Landsman, D., Lipman, D.J., Lu, Z., Madden, T.L., Madej, T., Maglott, D.R., Marchler-Bauer, A., Miller, V., Mizrachi, I., Ostell, J., Panchenko, A., Phan, L., Pruitt, K.D., Schuler, G.D., Sequeira, E., Sherry, S.T., Shumway, M., Sirotkin, K., Slotta, D., Souvorov, A., Starchenko, G., Tatusova, T.A., Wagner, L., Wang, Y., Wilbur, W.J., Yaschenko, E., Ye, J.: Database resources of the National Center for Biotechnology Information. Nucleic Acids Res **39**(Database issue) (2011) D38–51

[2] Baumbach, J.: On the power and limits of evolutionary conservation–unraveling bacterial gene regulatory networks. Nucleic Acids Res **38**(22) (2010) 7877–84

[3] del Toro, N., Dumousseau, M., Orchard, S.E., Jimenez, R.C., Galeota, E., Launay, G., Goll, J., Breuer, K., Ono, K., Salwínski, L., Hermjakob, H.: A new reference implementation of the psicquic web service. Nucleic Acids Research **41**(Webserver-Issue) (2013) 601–606

[4] Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R.M., Ideker, T.: Conserved patterns of protein interaction in multiple species. PNAS **102**(6) (2005) 1974–1979

[5] Memišević, V., Pržulj, N.: C-GRAAL: Common-neighbors-based global GRAph ALignment of biological networks. Integrative Biology **4** (2012) 734–743

[6] Heath, A.P., Kavraki, L.E.: Computational challenges in systems biology. Computer Science Review **3**(1) (2009) 1–17

[7] Atias, N., Sharan, R.: Comparative analysis of protein networks: hard problems, practical solutions. Commun. ACM **55**(5) (2012) 88–97

[8] Hart, G.T., Ramani, A., Marcotte, E.: How complete are current yeast and human protein-interaction networks? Genome Biology **7**(11) (2006) 120

[9] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of molecular biology **215**(3) (1990) 403–410

[10] Krissinel, E.B.: On the relationship between sequence and structure similarities in proteomics. Bioinformatics **23**(6) (2007) 717–723

[11] Tsai, W.H., Fu, K.S.: Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Transactions on Systems, Man, and Cybernetics **9**(12) (1979) 757–768

[12] Baumbach, J., Guo, J., Ibragimov, R.: Covering tree with stars. In Du, D.Z., Zhang, G., eds.: COCOON. Volume 7936 of Lecture Notes in Computer Science., Springer (2013) 373–384

[13] Baumbach, J., Guo, J., Ibragimov, R.: Covering tree with stars. Journal of Combinatorial Optimization ((accepted))

[14] Baumbach, J., Guo, J., Ibragimov, R.: Neighborhood-preserving mapping between trees. In Dehne, F., Solis-Oba, R., Sack, J.R., eds.: WADS. Volume 8037 of Lecture Notes in Computer Science., Springer (2013) 427–438

[15] Baumbach, J., Guo, J., Ibragimov, R.: Compactness-preserving mapping between trees. In Kulikov, A., Kuznetsov, S., Pevzner, P., eds.: CPM. Volume 8486 of Lecture Notes in Computer Science., Springer (2014) 162–171

[16] Ibragimov, R., Martens, J., Guo, J., Baumbach, J.: Nabeeco: biological network alignment with bee colony optimization algorithm. In Blum, C., Alba, E., eds.: GECCO (Companion), ACM (2013) 43–44

[17] Ibragimov, R., Malek, M., Guo, J., Baumbach, J.: Gedevo: An evolutionary graph edit distance algorithm for biological network alignment. In Beißbarth, T., Kollmar, M., Leha, A., Morgenstern, B., Schultz, A.K., Waack, S., Wingender, E., eds.: GCB. Volume 34 of OASICS., Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013) 68–79

[18] Ibragimov, R., Malek, M., Baumbach, J., Guo, J.: Multiple graph edit distance: simultaneous topological alignment of multiple protein-protein interaction networks with an evolutionary algorithm. In: GECCO. (2014) 277–284

[19] Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty Years Of Graph Matching In Pattern Recognition. IJPRAI **18**(3) (2004) 265–298

[20] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)

[21] Babai, L., Luks, E.M.: Canonical labeling of graphs. In Johnson, D.S., Fagin, R., Fredman, M.L., Harel, D., Karp, R.M., Lynch, N.A., Papadimitriou, C.H., Rivest, R.L., Ruzzo, W.L., Seiferas, J.I., eds.: STOC, ACM (1983) 171–183

[22] Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Systems Science and Cybernetics **4**(2) (1968) 100–107

[23] Justice, D., Hero, A.O.: A binary linear programming formulation of the graph edit distance. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8) (2006) 1200–1214

[24] Almohamad, H.A., Duffuaa, S.O.: A linear programming approach for the weighted graph matching problem. IEEE Trans. Pattern Anal. Mach. Intell. **15**(5) (1993) 522–525

[25] Zeng, Z., Tung, A.K.H., Wang, J., Feng, J., Zhou, L.: Comparing stars: On approximating graph edit distance. PVLDB **2**(1) (2009) 25–36

[26] Lin, C.L.: Hardness of approximating graph transformation problem. In Du, D.Z., Zhang, X.S., eds.: ISAAC. Volume 834 of Lecture Notes in Computer Science., Springer (1994) 74–82

[27] Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1) (2010) 113–129

[28] Matoušek, J., Thomas, R.: On the complexity of finding iso- and other morphisms for partial k-trees. Discrete Mathematics **108**(1-3) (1992) 343–364

[29] Marx, D., Pilipczuk, M.: Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In Mayr, E.W., Portier, N., eds.: STACS. Volume 25 of LIPIcs., Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2014) 542–553

[30] Arvind, V., Köbler, J., Kuhnert, S., Vasudev, Y.: Approximate graph isomorphism. In Rovan, B., Sassone, V., Widmayer, P., eds.: MFCS. Volume 7464 of Lecture Notes in Computer Science., Springer (2012) 100–111

[31] Arora, S., Frieze, A.M., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. Math. Program. **92**(1) (2002) 1–36

[32] Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters **18**(8) (1997) 689–694

[33] Matula, D.W.: Subtree isomorphism in $o(n^{5/2})$. Annals of Discrete Mathematics **2** (1978) 91–106

[34] Chung, M.J.: $o(n^(2.55))$ time algorithms for the subgraph homeomorphism problem on trees. J. Algorithms **8**(1) (1987) 106–112

[35] Shamir, R., Tsur, D.: Faster subtree isomorphism. J. Algorithms **33**(2) (1999) 267–280

[36] Hopcroft, J.E., Tarjan, R.E.: Isomorphism of planar graphs. In Miller, R.E., Thatcher, J.W., eds.: Complexity of Computer Computations. The IBM Research Symposia Series, Plenum Press, New York (1972) 131–152

[37] Tai, K.C.: The tree-to-tree correction problem. J. ACM **26**(3) (1979) 422–433

[38] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. **18**(6) (1989) 1245–1262

[39] Klein, P.N.: Computing the edit-distance between unrooted ordered trees. In Bilardi, G., Italiano, G.F., Pietracaprina, A., Pucci, G., eds.: ESA. Volume 1461 of Lecture Notes in Computer Science., Springer (1998) 91–102

[40] Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. ACM Transactions on Algorithms **6**(1) (2009)

[41] Pawlik, M., Augsten, N.: Rted: A robust algorithm for the tree edit distance. PVLDB **5**(4) (2011) 334–345

[42] Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Inf. Process. Lett. **42**(3) (1992) 133–139

[43] Zhang, K., Jiang, T.: Some max snp-hard results concerning unordered labeled trees. Inf. Process. Lett. **49**(5) (1994) 249–254

[44] Shasha, D., Wang, J.T.L., Zhang, K., Shih, F.Y.: Exact and approximate algorithms for unordered tree matching. IEEE Transactions on Systems, Man, and Cybernetics **24**(4) (1994) 668–678

[45] Akutsu, T., Fukagawa, D., Takasu, A.: Approximating tree edit distance through string edit distance. Algorithmica **57**(2) (2010) 325–348

[46] Akutsu, T., Fukagawa, D., Takasu, A., Tamura, T.: Exact algorithms for computing the tree edit distance between unordered trees. Theor. Comput. Sci. **412**(4-5) (2011) 352–364

[47] Akutsu, T., Tamura, T., Fukagawa, D., Takasu, A.: Efficient exponential time algorithms for edit distance between unordered trees. In Kärkkäinen, J., Stoye, J., eds.: CPM. Volume 7354 of Lecture Notes in Computer Science., Springer (2012) 360–372

[48] Akutsu, T., Fukagawa, D., Halldórsson, M.M., Takasu, A., Tanaka, K.: Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees. Theor. Comput. Sci. **470** (2013) 10–22

[49] Bille, P.: A survey on tree edit distance and related problems. Theor. Comput. Sci. **337**(1-3) (2005) 217–239

[50] Tahraoui, M.A., Pinel-Sauvagnat, K., Laitang, C., Boughanem, M., Kheddouci, H., Ning, L.: A survey on tree matching and xml retrieval. Computer Science Review **8** (2013) 1–23

[51] Yannakakis, M.: Node- and edge-deletion np-complete problems. [119] 253–264

[52] Lewis, J.M., Yannakakis, M.: The node-deletion problem for hereditary properties is np-complete. J. Comput. Syst. Sci. **20**(2) (1980) 219–230

[53] Burzyn, P., Bonomo, F., Durán, G.: Np-completeness results for edge modification problems. Discrete Applied Mathematics **154**(13) (2006) 1824–1844

[54] Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. Discrete Applied Mathematics **113**(1) (2001) 109–128

[55] Kirkpatrick, D.G., Hell, P.: On the completeness of a generalized matching problem. [119] 240–245

[56] Edmonds, J.: Paths, trees, and flowers. Canadian Journal of mathematics **17**(3) (1965) 449–467

[57] Mucha, M., Sankowski, P.: Maximum matchings via gaussian elimination. In: FOCS, IEEE Computer Society (2004) 248–255

[58] Gavril, F., Itai, A.: Covering a tree by a forest. In Lipshteyn, M., Levit, V.E., McConnell, R.M., eds.: Graph Theory, Computational Intelligence and Thought. Volume 5420 of Lecture Notes in Computer Science., Springer (2009) 66–76

[59] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)

[60] Feng, Q., Wang, J., Chen, J.: Matching and p 2-packing: Weighted versions. In Fu, B., Du, D.Z., eds.: COCOON. Volume 6842 of Lecture Notes in Computer Science., Springer (2011) 343–353

[61] Kelley, B.P., Yuan, B., Lewitter, F., Sharan, R., Stockwell, B.R., Ideker, T.: PathBLAST: a tool for alignment of protein interaction networks. Nucleic Acids Research **32**(Web-Server-Issue) (2004) 83–88

[62] Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R.M., Ideker, T.: Conserved patterns of protein interaction in multiple species. Proceedings of the National Academy of Sciences of the United States of America **102**(6) (2005) 1974–1979

[63] Kalaev, M., Smoot, M.E., Ideker, T., Sharan, R.: NetworkBLAST: comparative analysis of protein networks. Bioinformatics **24**(4) (2008) 594–596

[64] Koyutürk, M., Kim, Y., Topkara, U., Subramaniam, S., Szpankowski, W., Grama, A.: Pairwise Alignment of Protein Interaction Networks. Journal of Computational Biology **13**(2) (2006) 182–199

[65] Flannick, J., Novak, A., Srinivasan, B.S., McAdams, H.H., Batzoglou, S.: Graemlin: General and robust alignment of multiple large interaction networks. Genome Research **16**(9) (2006) 1169–1181

[66] Kuchaiev, O., Pržulj, N.: Integrative network alignment reveals large regions of global network similarity in yeast and human. Bioinformatics **27**(10) (2011) 1390–1396

[67] Singh, R., Xu, J., Berger, B.: Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology. In Speed, T.P., Huang, H., eds.: RECOMB. Volume 4453 of Lecture Notes in Computer Science., Springer (2007) 16–31

[68] Kuchaiev, O., Milenković, T., Memišević, V., Hayes, W., Pržulj, N.: Topological network alignment uncovers biological function and phylogeny. Journal of The Royal Society Interface **7** (2010) 1341–1354

[69] Milenković, T., Ng, W.L., Hayes, W., Pržulj, N.: Optimal network alignment with graphlet degree vectors. Cancer informatics **9** (2010) 121–137

[70] Aladag, A.E., Erten, C.: SPINAL: scalable protein interaction network alignment. Bioinformatics **29**(7) (2013) 917–924

[71] Patro, R., Kingsford, C.: Global network alignment using multiscale spectral signatures. Bioinformatics **28**(23) (2012) 3105–3114

[72] Zaslavskiy, M., Bach, F.R., Vert, J.P.: Global alignment of protein-protein interaction networks by graph matching methods. Bioinformatics **25**(12) (2009)

[73] Riesen, K., Neuhaus, M., Bunke, H.: Bipartite graph matching for computing the edit distance of graphs. In Escolano, F., Vento, M., eds.: GbRPR. Volume 4538 of Lecture Notes in Computer Science., Springer (2007) 1–12

[74] Chindelevitch, L., Liao, C.S., Berger, B.: Local Optimization for Global Alignment of Protein Interaction Networks. In Altman, R.B., Dunker, A.K., Hunter, L., Murray, T., Klein, T.E., eds.: Pacific Symposium on Biocomputing, World Scientific Publishing (2010) 123–132

[75] Chindelevitch, L., Ma, C.Y., Liao, C.S., Berger, B.: Optimizing a global alignment of protein interaction networks. Bioinformatics **29**(21) (2013) 2765–2773

[76] El-Kebir, M., Heringa, J., Klau, G.W.: Lagrangian Relaxation Applied to Sparse Global Network Alignment. In Loog, M., Wessels, L.F.A., Reinders, M.J.T., de Ridder, D., eds.: PRIB. Volume 7036 of Lecture Notes in Computer Science., Springer (2011) 225–236

[77] Klau, G.W.: A new graph-based method for pairwise global network alignment. BMC Bioinformatics **10**(S-1) (2009)

[78] Neyshabur, B., Khadem, A., Hashemifar, S., Arab, S.S.: NETAL: a new graph-based method for global alignment of protein-protein interaction networks. Bioinformatics **29**(13) (2013) 1654–1662

[79] Koyutürk, M., Kim, Y., Subramaniam, S., Szpankowski, W., Grama, A.: Detecting Conserved Interaction Patterns in Biological Networks. Journal of Computational Biology **13**(7) (2006) 1299–1322

[80] Kalaev, M., Bafna, V., Sharan, R.: Fast and Accurate Alignment of Multiple Protein Networks. Journal of Computational Biology **16**(8) (2009) 989–999

[81] Deniélou, Y.P., Boyer, F., Viari, A., Sagot, M.F.: Multiple Alignment of Biological Networks: A Flexible Approach. In: CPM. (2009) 263–273

[82] Boyer, F., Morgat, A., Labarre, L., Pothier, J., Viari, A.: Syntons, metabolons and interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. Bioinformatics **21**(23) (2005) 4209–4215

[83] Liao, C.S., Lu, K., Baym, M., Singh, R., Berger, B.: IsoRankN: spectral methods for global alignment of multiple protein networks. Bioinformatics **25**(12) (2009)

[84] Flannick, J., Novak, A.F., Do, C.B., Srinivasan, B.S., Batzoglou, S.: Automatic Parameter Learning for Multiple Local Network Alignment. Journal of Computational Biology **16**(8) (2009) 1001–1022

[85] Sahraeian, S.M.E., Yoon, B.J.: SMETANA: Accurate and Scalable Algorithm for Probabilistic Alignment of Large-Scale Biological Networks. PLoS ONE **8**(7) (07 2013) e67995

[86] Shih, Y.K., Parthasarathy, S.: Scalable global alignment for multiple biological networks. BMC Bioinformatics **13**(S-3) (2012) S11

[87] Alkan, F., Erten, C.: BEAMS: backbone extraction and merge strategy for the global many-to-many alignment of multiple PPI networks. Bioinformatics **30**(4) (2014) 531–539

[88] Hu, J., Kehr, B., Reinert, K.: NetCoffee: a fast and accurate global alignment approach to identify functionally conserved proteins in multiple networks. Bioinformatics **30**(4) (2014) 540–548

[89] Schlicker, A., Domingues, F.S., Rahnenführer, J., Lengauer, T.: A new measure for functional similarity of gene products based on Gene Ontology. BMC Bioinformatics **7** (2006) 302

[90] Mueller, L.A., Dehmer, M., Emmert-Streib, F.: Comparing Biological Networks: A Survey on Graph Classifying Techniques. In Prokop, A., Csukás, B., eds.: Systems Biology. Springer Netherlands (2013) 43–63

[91] Mohammadi, S., Grama, A.: Biological Network Alignment. In Koyutürk, M., Subramaniam, S., Grama, A., eds.: Functional Coherence of Molecular Networks in Bioinformatics. Springer New York (2012) 97–136

[92] Solé-Ribalta, A., Serratosa, F.: Graduated assignment algorithm for multiple graph matching based on a common labeling. IJPRAI **27**(1) (2013)

[93] Pržulj, N.: Biological network comparison using graphlet degree distribution. Bioinformatics **23**(2) (2007) 177–183

[94] Milenković, T., Pržlj, N.: Uncovering Biological Network Function via Graphlet Degree Signatures. Cancer Informatics **6** (2008) 257–273

[95] Xie, J., Zhang, S.H., Wen, T., Ding, G., Yu, S., Gu, Z., Zhang, W.: A Querying Method with Feedback Mechanism for Protein Interaction Network. In: HISB, IEEE (2011) 351–358

[96] Parrish, J., Yu, J., Liu, G., Hines, J., Chan, J., Mangiola, B., Zhang, H., Pacifico, S., Fotouhi, F., DiRita, V., Ideker, T., Andrews, P., Finley, R.: A proteome-wide protein interaction map for *Campylobacter jejuni*. Genome Biology **8**(7) (2007) R130

[97] Shimoda, Y., Shinpo, S., Kohara, M., Nakamura, Y., Tabata, S., Sato, S.: A Large Scale Analysis of Protein-Protein Interactions in the Nitrogen-fixing Bacterium *Mesorhizobium loti*. DNA Research **15**(1) (2008) 13–23

[98] Sato, S., Shimoda, Y., Muraki, A., Kohara, M., Nakamura, Y., Tabata, S.: A Large-scale Protein-protein Interaction Analysis in *Synechocystis sp. PCC6803*. DNA Research **14**(5) (2007) 207–216

[99] Peregrin-Alvarez, J.M., Xiong, X., Su, C., Parkinson, J.: The Modular Organization of Protein Interactions in *Escherichia coli*. PLoS Comput Biol **5**(10) (2009) e1000523

[100] Collins, S.R., Kemmeren, P., Zhao, X.C., Greenblatt, J.F., Spencer, F., Holstege, F.C.P., Weissman, J.S., Krogan, N.J.: Toward a Comprehensive Atlas of the Physical Interactome of Saccharomyces cerevisiae. Molecular and Cellular Proteomics **6**(3) (2007) 439–450

[101] Xenarios, I., Salwínski, L., Joyce, X., Higney, P., Kim, S.M.M., Eisenberg, D.: DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions. Nucleic acids research **30**(1) (2002) 303–305

[102] Ulitsky, I., Karp, R.M., Shamir, R.: Detecting Disease-Specific Dysregulated Pathways Via Analysis of Clinical Expression Profiles. In Vingron, M., Wong, L., eds.: RECOMB. Volume 4955 of Lecture Notes in Computer Science., Springer (2008) 347–359

[103] Radivojac, P., Peng, K., Clark, W.T., Peters, B.J., Mohan, A., Boyle, S.M., Mooney, S.D.: An integrated approach to inferring gene-disease associations in humans. Proteins: Structure, Function, and Bioinformatics **72**(3) (2008) 1030–1037

[104] Prasad, K.T.S., Goel, R., Kandasamy, K., Keerthikumar, S., Kumar, S., Mathivanan, S., Telikicherla, D., Raju, R., Shafreen, B., Venugopal, A.,

Balakrishnan, L., Marimuthu, A., Banerjee, S., Somanathan, D.S., Sebastian, A., Rani, S., Ray, S., Kishore, H.C.J., Kanth, S., Ahmed, M., Kashyap, M.K., Mohmood, R., Ramachandra, Y.L., Krishna, V., Rahiman, A.B., Mohan, S., Ranganathan, P., Ramabadran, S., Chaerkady, R., Pandey, A.: Human Protein Reference Database–2009 update. Nucleic acids research **37**(Database issue) (2009) D767–D772

[105] Karaboga, D., Akay, B.: A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation **214**(1) (2009) 108–132

[106] Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review (2012) 1–37

[107] Barabási, A.L., Albert, R.: Emergence of Scaling in Random Networks. Science **286**(5439) (1999) 509–512

[108] Eiben, A.E., Smith, J.E.: Introduction to evolutionary computing. Natural Computing Series. Springer (2010)

[109] Jong, K.A.D.: Evolutionary computation - a unified approach. MIT Press (2006)

[110] Liu, C.W., Fan, K.C., Horng, J.T., Wang, Y.K.: Solving weighted graph matching problem by modified microgenetic algorithm. In: Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on. Volume 1., IEEE (1995) 638–643

[111] Cross, A.D.J., Wilson, R.C., Hancock, E.R.: Inexact graph matching using genetic search. Pattern Recognition **30**(6) (1997) 953–970

[112] Bärecke, T., Detyniecki, M.: Memetic algorithms for inexact graph matching. In: IEEE Congress on Evolutionary Computation, IEEE (2007) 4238–4245

[113] Goldberg, D.E., Linge, R.J.: Alleles, loci, and the traveling salesman problem. In Grefenstette, J.J., ed.: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Lawrence Erlbaum Associates, Publishers (1985)

[114] Bayati, M., Gleich, D.F., Saberi, A., Wang, Y.: Message-Passing Algorithms for Sparse Network Alignment. TKDD **7**(1) (2013) 3

[115] Wang, B., Gao, L.: Seed selection strategy in global network alignment without destroying the entire structures of functional modules. Proteome Science **10**(1) (2012)

[116] Li, C., Yang, S.: An island based hybrid evolutionary algorithm for optimization. In: SEAL. (2008) 180–189

[117] Wiwie, C., Baumbach, J., Röttger, R.: Standardization and Evaluation of Popular Bioinformatics Clustering Tools - An Integrated Online Framework. (in preparation) **-**(-) (2014) –

[118] Saito, R., Smoot, M.E., Ono, K., Ruscheinski, J., Wang, P.L., Lotia, S., Pico, A.R., Bader, G.D., Ideker, T.: A travel guide to Cytoscape plugins. Nature Methods **9**(11) (2012) 1069–1076

[119] Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V., eds.: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA. In Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V., eds.: STOC, ACM (1978)