FINDING OPTIMAL SPANNING K-TREES IN BACKBONE GRAPHS

by

Abdul Samad

(Under the Direction of Liming Cai)

Abstract

Many intractable problems on graphs are polynomial time solvable when the graphs have bounded treewidth. An important class of the graphs with bounded treewidth is called k-trees, which are formed by starting with a k-clique and then repeatedly adding vertices in such a way that each added vertex has exactly k neighbors that form a new clique. Finding spanning k-trees has applications in networks where it was first studied in networks, and was later shown to be NP-Complete even for k=2.

Biomolecule (protein, RNA) structure prediction is a grand challenge which leads to many computation methods in bioinformatics. We model the biomolecular structure prediction problem as finding the maximum spanning k-tree on the backbone graphs, where the backbone graphs are characterized by a linear sequence of the vertices. However, most of the traditional methods are not powerful to search through the huge conformation space. This implies a strong demand for more efficient models.

We prove that the problem is W[1]-hard for the objective function defined on the cliques of the input graph. For solving the problem, we show that the problem can be solved in time $O(k.(n)^{k+1}(k+1)^{k+2})$ for every given k and we give evidence that our algorithm is very likely to be optimal. Our algorithm also works for different objective function defined over the cliques of the k-tree, which enables pertinent characterizations of real world problems.

INDEX WORDS : K-tree, Tree Decomposition, Dynamic Programming, Optimal .

FINDING OPTIMAL SPANNING K-TREES IN BACKBONE GRAPHS

by

Abdul Samad

B.S.C.S Balochistan University of Information and Technology &

Management Sciences, Pakistan, 2006.

A Dissertation submitted to the graduate faculty

of the university of Georgia in partial fulfillment

of the

requirement for the degree

DOCTOR OF PHILOSOPHY

ATHENS, GA

2013

FINDING OPTIMAL SPANNING K-TREES IN BACKBONE GRAPHS

by

Abdul Samad

Major Professor: Liming Cai

Committee: Robert W.Robinson

E. Rodney Canfield

Russell L. Malmberg

Electronic version Approved:

Maureen Grasso

Dean Of Graduate School

The University of Georgia

August 2013

Dedication

This is dedicated to my wife, father and mother.

Acknowledgment

I am thankful to to my committee members. I wish to express my sincere gratitude to Liang Ding and Dr. Goujun Li for providing helpful tips.

Contents

Acknowledgment **v**

List of Figures **ix**

List of Symbols **1**

1 Introduction **1**

1.1 Motivation 2

1.2 Contribution of the Dissertation 3

2 Preliminaries **5**

2.1 Graph Theory 5

2.2 Properties of k-trees 7

2.3 Maximum Spanning k-trees 9

3 Properties of k-trees **12**

4 Algorithms for MskT and MskT-C **17**

4.1 2-trees 18

4.2 Finding Optimal k-trees for k=3 28

4.3 Algorithm Analysis 47

5 Hardness **53**

6 K-trees as Stochastic Context Sensitive Grammars **56**

6.1 Relationship of SCSGs with Backbone k-trees 57

6.1.1 Faithful k-trees and backbone k-trees 58

6.2 CSGs for 2-trees 60

6.3 SCS Language Models 61

6.3.0.1 Probability distributions of strings 61

6.3.0.2 Algorithms 65

Conclusion **68**

References **69**

List of Figures

4.1 2-tree 20

4.2 $P\{x_i, x_k, x_k\}$ 22

4.3 $L\{x_i, x_k, x_k\}$ 23

4.4 $U\{x_i, x_k, x_k\}$ 24

4.5 2-children 29

4.6 3-children 30

4.7 4-branches 30

4.8 $\mathbf{M_{(1,1,1,1,r=1)}}(x_i, x_j, x_m)$ 31

4.9 $\mathbf{M_{(1,0,0,0,r=1)}}(x_i, x_j, x_m)$ 35

4.10 $\mathbf{M_{(0,1,0,0,r=2)}}(x_i, x_j, x_m)$ 36

4.11 $\mathbf{M_{(0,0,1,0,r=3)}}(x_i, x_j, x_m)$ 37

4.12 $\mathbf{M_{(0,0,0,1,r=4)}}(x_i, x_j, x_m)$ 38

4.13 $\mathbf{M_{(1,1,0,0,r=1)}}\{x_i, x_j, x_m\}$ 39

4.14 $\mathbf{M_{(1,1,0,0,r=2)}}\{x_i, x_j, x_m\}$ 40

4.15 $\mathbf{M_{(0,1,1,0,r=2)}}\{x_i, x_j, x_m\}$ 41

4.16 $\mathbf{M_{(1,0,1,0,r=1)}}\{x_i, x_j, x_m\}$ 42

4.17 $\mathbf{M_{(1,0,0,1,r=1)}}\{x_i, x_j, x_m\}$ 43

4.18 $\mathbf{M_{(1,1,1,0,r=1)}}\{x_i, x_j, x_m\}$ 45

4.19 illustrates (a) duplicate siblings in a k-tree, and (b) transformation of the k-tree to one without duplicate sibling 50

6.1 (a) "Parallel" and "nested" relationships between letters on string `abcacbacafadeaeda` of length 17, characterizable with SCFG; (b) "crossing" relationships as the result of including additional (red color) relationships, which cannot be modeled with SCFG. 57

6.2 (a) A 3-tree faithful to a string of length 7, (b) after vertex 4 is added to the k-tree in (a), it becomes a backbone 3-tree for the string, and (c) a 3-tree but not faithful to the string. Bold indicates backbone edge. 59

6.3 One possible syntactic structure of 2-tree for string $ACGUAC$ 61

6.4 (a) Illustration for generation of a 3-tree with 7 vertices (b) Tree topology representation of the 3-tree in (a). 62

Chapter 1

Introduction

It has been observed that many important intractable problems on graphs are polynomial time solvable when the graphs are constrained to having small treewidth [40]. The notion of treewidth, introduced by Robertson and Seymour [34], has many important implications in algorithm design and computational complexity, resulting efficient algorithms that can recognize graphs of bounded treewidth, determine bounds of treewidth for various classes of graphs, and construct graph tree decompositions [32, 33]. An important class of constrained graphs is the class of k-trees, where integer $k \geq 1$, which can be defined recursively as follows: (1) A k-clique is a k-tree of k-vertices; and (2) connecting a new vertex to any k-clique in an existing k-tree of n vertices forms a k-tree of $n+1$ vertices. When $k = 1$, a k-tree is a tree in the general sense. Since the treewidth of a k-tree is bounded by k, many NP-hard problems are solvable on k-trees in time $O(f(k)P(n))$, where $P(n)$ is a polynomial the input size n and $f(k)$ is some function, often exponential such as $2^{O(k)}$ [39, 37, 38, 36, 35].

An early application for k-tree was the design of reliable networks, for instance minimal IFI networks are exactly 2-trees [47, 48]. It was shown [12] that deciding whether a graph contains a spanning k-tree is NP-complete, for each fixed $k \geq 2$. The problem remains NP-complete for restricted classes of graphs such as split graphs, graphs with maximum degree $3k + 2$, or planar graphs with maximum degree 6 [13].

1.1 Motivation

Biological macromolecules such as protein or RNA are building blocks of an organism. Knowing the structure of such a molecules is important for understanding it's function [15] which may allow us to comprehend roles of different biomolecules and thus overall biological process. Such knowledge is important, for example in drug discovery, to design new molecules with desirable functions [1, 2].

Experimentally determining the structure of a protein or RNA using NMR and X-ray crystallography is a complicated and laborious task which is time-consuming and may not yield the structures with high throughput processes [3]. Computational determination of biomolecule structure based on the sequence of residues, and possibly other easily obtainable information, is becoming useful and successful in a number of biomecule research [4]. However, often such structure prediction problems are computationally intractable [6, 7, 8, 5, 11].

A single stranded biomolecule is a linear sequence $X = x_1x_2...x_n$ of length n consist of n residues $x_i \in \Sigma$; the sequence is also called primary structure of the molecule. For RNA the alphabet is $\Sigma = \{A, C, G, U\}$, and for proteins the alphabet is the set of 20 amino acids. The interactions due to chemical bonds among the residues cause the molecule sequence to fold back onto itself [9, 10]. Folding causes two higher level structures, the secondary structure that is due to local interactions and tertiary structure because of the additional global interactions. This dissertation was first motived by the task to predict the structure of RNA given the primary structure as input. However, the work applies to prediction of proteins structure as well.

In the RNA secondary structure, canonical base pairs between residues, such as the Watson-Crick base pairs (AU and GC) and wobble pair GU, almost always occur in the nested fashion [46]. Non-nested base pairs give rise to pseudoknots. Residue interactions consume energy; a secondary structure with the minimum free energy is the most stable structure for a given sequence. For an RNA sequence of 200 base long, there are over 10^{50} possible base paired structures [50]. Since not all of these structures are biologically relevant, the biologically correct structures need to be distinguished from incorrect ones. Computationally, this is done by introducing an objective a function which assigns higher scores to biologically more relevant structures. Desirable algorithms are sought to evaluate and identify the maximum score corresponding to the most plausible structure.

There have been a number of algorithms developed for RNA secondary structure prediction. For example, Nussinov's folding algorithm uses recursion to calculate the best structure for every smaller sub-sequences, and works towards larger sub-sequences [16]. To achieve the fold stability, Nussinov's algorithm computes to maximize the total number of base-pairs. Zuker *et al.* [17] proposed an extended approach in which the free energy of secondary structure is calculated from the approximation of sums of contributions of the structural units such as stems and loops which are present in the structures. Current programs such as $Mfold$ [19] and $RNAfold$ [18] all use this strategy.

Programs such as $PFold$ [26, 21] and $CONTRAfold$ [24] have been developed from the probabilistic models for RNA structure [27, 23]. Stochastic context free grammars (SCFGs) have been used to model the secondary structure of RNA. An SCFG is formed from a CFG by associating a probability distribution with the productions of each nonterminal. Compounding the probabilities of rules used in a specific generation process for a string gives rise to the probability for the specific syntactic structure admitted by the string. Computing the secondary structure with the maximum probability can be accomplished by well-known CYK algorithm [25, 22]. However, these mentioned algorithms do not consider the psuedoknots.

1.2 Contribution of the Dissertation

This dissertation proposes a new algorithm for finding optimal spanning k-trees on given constrained graph which can also be used for *de novo* structure prediction for biomolecules (*i.e.,* RNA and protein). The research was motivated by the investigation of the interaction topology graph model for biomolecules [14]. In this model, the vertices in the graph represent nucleotides and edges represent potential interactions among nucleotides. The problem of RNA structure prediction for a given RNA molecule sequence is thus formulated as finding an optimal spanning k-tree from the corresponding interaction topology graph. This is based on the observation that most of the RNA structures have small treewidth [49].

In the second chapter we will define some graph theory concepts which will be used in the remaining chapters. We will formally define backbone graphs to be graphs containing the primary structure. We will pose the optimization problem as that of finding a spanning k-tree containing the backbone while maximizing an objective function of a general form.

In Chapter 3 we discuss the properties of spanning k-trees which contains the backbone edges. We will specialize the discussion to the 2-trees ($k = 2$). A $O(n^3)$ time algorithm will be presented for finding an optimal spanning 2-tree for a given graph.

In chapter 4 the algorithm for 2-tree is generalized and extended for k-trees. Complete details are given for 3-trees, and a framework is presented for $k > 3$. An algorithm for finding a optimal spanning k-tree is given for given graph $G = (V, E)$ and an integer k, with running time $O(n^{k+1}(k+1)^{k+2}k)$.

In chapter 5 we will discuss the hardness results for spanning k-tree problem. W[1] hardness result of the problem is also discussed.

In chapter 6 a new stochastic model is introduced for processing a mildly context sensitive language. The modeled structures are k-trees, which can be defined with recursive rules. We are going to associate a probability distribution with syntactic rules and use the algorithm developed in the chapter 4 for statistical analysis with k-tree structure.

Chapter 2

Preliminaries

2.1 Graph Theory

In this section, some of the basic notation of graph theory is discussed. Graphs are mathematical structures which are used to model pairwise relations between objects. They have proven to be an effective modeling tool in many disciplines, where vertices in the graph model are objects in the problem domain, and the edges between the vertices are used to model the corresponding interactions among the objects.

Definition 2.1.1. A graph G is defined as tuple (V, E), where V is the set of vertices and E is the set of edges. An edge $e = \{u, v\}$ is an unordered pair of distinct vertices; thus the graph we consider are always simple (no loops or multiple edges) and undirected.

Given an edge $e = \{u, v\}$ we say that u and v are the *endpoints* of e. A vertex is *incident* to an edge if it is one of the endpoints of that edge, so for any given two distinct vertices in a graph, we say that they are *adjacent* if they are both incident to a common edge. If there is an undirected edge $\{u, v\}$ in G, we say that u and v are adjacent. Often $\{u, v\}$ is abbreviated as uv. The vertices adjacent to a vertex u are called its neighbors. A *path* in G is a non-empty graph $P = (V', E')$, $V' \subseteq V$ and $E' \subseteq E$, where $V' = \{v_1, v_2, \ldots, v_k\}$, $E' = \{v_1v_2, v_2v_3, \ldots, v_{k-1}v_k\}$ and the v_i are all distinct. A *cycle* $C = (V', E')$ is defined similarly with the exception that $v_1 = v_k$. Graph G is called *connected* if there is path between every pair of vertices in G. A graph is said to be *disconnected* if it is not connected. A *tree* T is a connected graph with no cycles. In a *rooted* tree one of the vertices is distinguished as the *root*. In a rooted tree, for any two neighboring vertices the one closer to the root is called the *parent* and the other is called the *child*.

Given a graph G, a vertex $v \in V$, and an edge $e \in E$, then $G - v$ denotes the graph $(V \setminus \{v\},\, E \setminus \{e \in E : e\,\text{is incident to}\,v\})$ and $G - e$ denotes the graph $(V, E \setminus \{e\})$. These two operations are called deleting a vertex and deleting an edge, respectively. Any graph that can be obtained via these two operations is a *subgraph* of G. If all the vertices of the graph $G = (V,\, E)$ are pairwise adjacent, then G is called *complete* and is denoted by K^n where n is the number of vertices. *E.g.*, K^2 is an edge and K^3 is a triangle. A clique is a complete subgraph. A subgraph $G' = (V',\, E')$ of $G = (V,\, E)$ is called induced if E' contains all of the edges in E which have both endpoints in V'. A set $S \subset V$ is a separator of of the graph $G = (V,\, E)$ if the subgraph of G induced by $V - S$ is disconnected. The set S is a uv-separator if u and v are in different connected components of subgraph of G induced by $V - S$. A uv separator S is minimal if no subset of S separates u and v. The book by Diestel [51] is a good source of further terminology and information about the graph theory.

It is often the case that a problem which is intractable for general graphs becomes easy to solve when restricted to graphs with simple structures, such as trees. The reason can be intuited in the following way: Let T be rooted tree and T_u denote the subtree induced by u and its descendants. Then if $v_1,\, v_2, \ldots,\, v_k$ are the children of u, the solution of T_u is obtained from solutions on $T_{v_1},\, T_{v_2}, \ldots,\, T_{v_k}$ considering how they interact at u. Many NP-hard graph problems can be solved efficiently when the underlying graph is restricted to be a tree or tree-like structure. The notion of *Tree-decomposition* was introduced to view sparse graphs as tree like structures.

Definition 2.1.2. A tree-decomposition of a graph $G = (V,\, E)$ is a pair

$$(\{X_i | i \in I\}, T = (I, F))$$

where $\{X_i | i \in I\}$ is a family of subsets of V, one for each node of T, and T is a tree such that

1. $\bigcup_{i \in I} X_i = V$;
2. For each edges $\{u, w\} \in E$, there is an $i \in I$ with $u \in X_i$ and $w \in X_i$;
3. For all i, j, $k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

The *treewidth* of a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is $max_{i \in I} |X_i| - 1$. The treewidth of graph G is the minimum treewidth over all possible tree-decompositions of G [52]. Computing the treewidth and finding a corresponding tree-decomposition of a graph is an NP-hard problem [55].

Research in algorithmic graph theory has revealed that many hard problems are easy when restricted to graphs of small treewidth. This applies to classes of graphs with the property that there is an upper bound on the tree width of the graphs in that class. Examples of such classes are trees (treewidth$\leq$1), series parallel networks (treewidth$\leq$2) and k-trees (treewidth k) [54]. We will restrict our discussion from now on to k-trees.

2.2 Properties of k-trees

Definition 2.2.1. A *k-tree* is a graph that can be generated via the following rules [53].

1. A *k-clique* is a k-tree of k vertices.
2. A k-tree G on $n + 1$ vertices ($n \geq k$) can be obtained from a k-tree H with n vertices by adding a new vertex and k edges which join it to some k-clique in H.

Definition 2.2.2. A *partial k-tree* is a subgraph of a k-tree.

Theorem 2.2.3. [55] A G is a partial k-tree if and only if G has tree width at most k.

Theorem 2.2.4. [13] Let k be a positive integer, T be a k-tree with n vertices, and v be a vertex of T. Then the following are true;

- T contains no chordless cycle of length at least 4,
- T contains no clique of size $(k+2)$,
- T has exactly $kn - \binom{k+1}{2}$ edges,
- T has exactly $n-k$ cliques of size $(k+1)$,
- either T is a k-clique or every maximal clique of T is of size $(k+1)$, and
- the neighborhood of any vertex of T is $(k-1)$-tree.

An early work on k-trees [53] characterized k-trees in terms of disconnection properties and forbidden subgraphs.

Theorem 2.2.5. A graph $G = (V, E)$ is a k-tree if and only if every minimal xy-separator of G is k-clique.

Assume $G = (V, E)$ is connected graph. Let S be a separator of G such that the induced graph $G(V-S)$ has two or more connected components, say $C_i = (V_i, E_i)$. Then subgraphs $G(V_i \cup S)$ are the leaves of G with respect to S [53].

Note: We will use the term branches as compared to leaves. Because leave is standard term in graph theory used for the vertices with degree 1.

Theorem 2.2.6. [53] Let $G = (V, E)$ be a k-tree with S a separation clique. Then each branch of G with respect to S is a k-tree.

2.3 Maximum Spanning k-trees

From now any graph $G = (V,\ E)$ on n vertices is assumed to have vertex set $V = \{x_1,\ x_2, \ldots, x_n\}$ or simply $\{1,\ 2, \cdots,\ n\}$ for convenience.

Definition 2.3.1. Let $G = (V,\ E)$ be a graph. A *spanning* k-tree of G is a k-tree T, $T = (V,\ E^{'})$, with vertex set V and edge set $E^{'} \subseteq E$.

There may be many spanning k-trees for a given graph. We can define various objective functions over a spanning k-tree. The most straightforward objective function is the weight sum of its edges. The problem to maximize this objective function can be expressed as follows.

Problem: maximum spanning k-tree (MskT)

Input: Graph $G = (V,\ E)$ with weight function $w\ : E \to R_{\geq 0}$, and integer $k \geq 1$.

Output: A spanning k-tree which maximizes the sum of its edge weights.

The minimization version of this problem was suggested by Farley [47], and it was shown by Bern [12] that the decision version of determining existence of a spanning k-tree is NP-complete for each fixed $k \geq 2$, for general graphs. We consider the maximum spanning k-tree problem restricted to graphs and spanning k-trees which contain a specified Hamiltonian path, called a backbone.

Definition 2.3.2. In an any graph $G = (V,\ E)$ with $V = \{x_1,\ x_2, ...,\ x_n\}$, the *backbone* is the edge set $B = \{x_1x_2,\ x_2x_3, \ldots,\ x_{n-1}x_n\}$ or else the graph, $(V,\ B)$. If $B \subseteq E$ then G is said to be a backbone graph or graph with backbone.

For our maximization problem the most general objective function considered is determined by a user defined non-negative weighting of the cliques. The weight for a graph is the sum of the weights of the cliques.

Definition 2.3.3. Let *clique*(G) denote the set of non-trivial cliques in G and w a weighting

$$w : \ clique(G) \rightarrow R_{\geq 0}$$

then w is extended to an arbitrary subgraph H of G by setting

$$\hat{w}(H) = \sum_{k \in clique(H)} w(k)$$

General maximum spanning k-trees with backbone (MskT-G)

Input: A complete backbone graph $G = (V, E)$ with weight function w, integer $k \geq 1$ and objective function, where $w : \ clique(G) \rightarrow R_{\geq 0}$.

Output: A backbone spanning k-tree, T such that $\hat{w}(T)$ is maximized w.r.t objective function.

For convenience we will still call a problem MskT even it when it is restricted to backbone graphs, as follows.

Maximum spanning k-trees edge-weighted with backbone (MskT)

Input: A complete backbone graph $G = (V, E)$ with weight function w and integer $k \geq 1$, where $w : \ E \rightarrow R_{\geq 0}$.

Output: A backbone spanning k-tree, $T = (V, F)$, $F \subseteq E$, such that $\sum_{(x,y) \in F} w(x, y)$ is maximized.

In many applications, the weights defined on the cliques will be more interesting as the weight function on the $(k+1)$-clique may not simply be replaced with the sum of edge weights in the clique. This is particularly true in the bio-molecule structure modeling for which interaction energy function tend to be multi-bodies instead of binary. If $clique(G)$ denote only the set of $(k+1)$-clique in G , we get another restricted version of the problem expressed as follows

Maximum spanning backbone k-trees on weighted (k+1)-cliques (MskT-C)

Input: A complete backbone graph $G = (V, E)$ with weight function w and integer $k \geq 1$, where $w : clique(G)\text{-clique} \rightarrow R_{\geq 0}$.

Output: A backbone spanning k-tree, $T = (V, F)$, $F \subseteq E$, such that $\sum_{\tau \in T} w(\tau)$ is maximized, where τ is $(k+1)$-clique.

Since every spanning k-tree needs to include all the backbone edges, the total weight of the backbone edges is invariant for the given graph when the objective is to maximize the edge weight sum.

Proposition 2.3.4. In the computation of MskT, we can ignore the weights of B.

Chapter 3

Properties of k-trees

In this chapter we will discuss some of the properties of spanning k-trees, especially on backbone graphs, which will facilitate discussion of algorithms for the problems MskT and MskT-C.

We first introduce a representation for k-trees. From [53] we know that a k-tree $T = (V, E)$ cannot contain a $(k+2)$-clique. An *addition sequence* for a k-tree T with $n > k$ vertices is a sequence of a $(k+1)$-cliques

$$S =< \kappa_0, \kappa_1, \ldots, \kappa_m > (m = n - k - 1)$$

where κ_0 is the called *base clique* of the sequence and for each $j > 0$, κ_j is created through replacing some vertex in some earlier $(k+1)$-clique by the new vertex [12]. That is, for each $j = 1, 2, ..., m$, the clique κ_j is defined as

$$\kappa_j = \kappa_i \setminus \{x\} \cup \{y\}$$

for some $i < j$ such that $x \in \kappa_i$ and $y \notin \kappa_l, \forall\, l < j$. This relationship will be dented by $\kappa_j = \kappa_i|^x_y$.

Proposition 3.0 We define a rooted tree topology $R(S)$ for a k-tree T from an addition sequence S for T by taking, for each $j > 0$, the parent of κ_j to be κ_i for the largest i, $i < j$, such that $\kappa_j = \kappa_i|^x_y$ for some x and y. call κ_j a *child* of κ_i and κ_i the *parent* of κ_j.

We also define a *descendent* of κ_i to be a child of κ_i or, recursively, a child of a descendent of κ_i. Let $U_{\kappa_i} = \{v \in \kappa : \kappa$ is a descendent of $\kappa_i\} \cup \kappa_i$, *i.e.*, the set of all vertices contained in κ_i and its descendants. For the root clique κ_0, $U\kappa_0 = \{1, 2, ..., n\}$. A special case of this clique sequence is a *k-path* where $i = j - 1$ whenever $\kappa_j = \kappa_i|_y^x$.

Proposition 3.1. For κ_i, κ_j $\in$ addition sequence for a k-tree, $\kappa_j = \kappa_i|_y^x$, where $i < j$ then the following properties hold

1. $U_{\kappa_j} \subset U_{\kappa_i}$;
2. $y \in U_{\kappa_i}$ and $x \notin U_{\kappa_j}$;
3. for every child clique κ_l of κ_i, if $l \neq j$, then $y \notin U_{\kappa_l}$.

Definition 3.2 Children cliques of the same parent are called *duplicate sibings* if they have a k-clique in common.

Note that due to Proposition 3.1 there will be no duplicate siblings in $R(S)$ when S is an addition sequence for a k-tree.

Proposition 3.3. Let $S =< \kappa_0, \kappa_1, ..., \kappa_m >$ be an addition sequence for a k-tree $G = (V, E)$. Then

1. $\forall v \in V$, $v \in \kappa$ for some clique κ in S.
2. $\forall (u, v) \in E, (u, v) \in \kappa$ some clique κ in S.
3. For any pair of cliques κ_i and κ_j in the S, $\kappa_i \cap \kappa_j \subseteq \kappa$ for every clique κ on the path between κ_i and κ_j in the tree induced by S.

Proof: (1) and (2) are obvious from the definition of a k-tree. For (3) by proposition 3.2 we know that S defines a tree topology T. In T there exists unique undirected path P between κ_i to κ_j and this path contains κ. For the contradiction we assume that there exists a vertex $x \in \kappa_i, \kappa_j$ but $x \notin \kappa$. Since κ lies on this P, U_κ will contain at least one of κ_i, κ_j assume $\kappa_i \subset U_\kappa$; so then there exists $\kappa_m \subset U_\kappa$ where x was first introduced but x is introduced in some κ_p where $\kappa_p \neq \kappa_m$ because $x \in \kappa_j$, violating the generation of k-tree.

Now we will derive some of the properties of k-trees on backbone graphs.

Definition 3.4. Let κ be a clique in a spanning k-tree with backbone on n vertices and $v \notin \kappa$. Then the *stretch* of v in κ is the maximal set of consecutive vertices on the backbone which contains v but none of the vertices in κ. We denote this set by stretch(κ, v) and call it a κ-*stretch*.

For example, if $\kappa = \{x_1, x_2, ..., x_{k+1}\}$ with $x_1 < x_2 < ... < x_{k+1}$, $v \notin \kappa$ and $x_i < v < x_{i+1}$ then for some i, stretch(κ, v) $= \{x_i + 1, \dots, v-1, v, v+1, \dots, x_{i+1} - 1\}$. We consider $x_0 = 0$ and $x_{k+2} = n + 1$ by convention in this context.

Proposition 3.5. For any $(k + 1)$-clique κ in a k-tree with backbone there are at most $(k + 2)$ non-empty κ-stretches.

Theorem 3.6. Let κ in an addition sequence S for a backbone k-tree $T = (V, E)$ and $v \in V$ and $v \notin \kappa$. Then stretch(κ, v) $\subset U_\kappa$ or stretch(κ, v) $\cap U_\kappa = \emptyset$.

Proof: Suppose stretch(κ, v) $\cap U_\kappa \neq \emptyset$. We will show that stretch(κ, v) $\subset U_\kappa$. For some vertex w we have $w \in$ stretch(κ, v)$\cap U_\kappa$. We claim that any vertex $w' \notin \kappa$ which is consecutive with w is also in U_k. Then $\{w, w'\}$ in T, since it is a backbone edge, so there must be a clique κ' such that $\{w, w'\} \in \kappa'$. If κ' is a descendant of κ then $w' \in U_\kappa$. So assume that κ' is not a descendant of κ. Since $w \in$ stretch(κ, v) $\cap U_\kappa$ we have that w is in some clique κ'' which is a descendant of κ. Then by Proposition 3.2, w must appear in every clique on the path from κ' to κ''. But κ must lie on this path, contradicting with the fact $w \notin \kappa$, so in fact $w' \in U_\kappa$ as claimed. Applying the claim inductively we find that stretch(κ, v) $\subset U_\kappa$ since stretch(κ, v) forms a path of consecutive vertices along the backbone.

Corollary 3.7. Let κ, κ' be in an addition sequence for a backbone k-tree such that $\kappa = \{x_1, x_2, ..., x_{k+1}\}$ with $x_1 < x_2 < ... < x_{k+1}$. If $\kappa' = \kappa|_y^{x_i}$, for some i and $y \notin \kappa$, $1 \leq i \leq k+1$ and $1 \leq y \leq$n, then

1. stretch(κ, y) $\subset U_{\kappa'}$;
2. stretch(κ', x_i) $\cap U_{\kappa'} = \emptyset$.

Definition 3.8. Let κ belong to an addition sequence for a k-tree. The *importable set* I_k, of κ is the set of vertices contained in the descendent cliques of κ, excluding those vertices that are already in κ, *i.e.*, $I_\kappa = U_\kappa \backslash \kappa$.

Proposition 3.9. Let κ' and κ'' are children of κ in the topology associated with a given addition sequence for a backbone k-tree. Then

1. If $\kappa' = \kappa|_y^x$ then $I_{\kappa'} = I_\kappa \setminus$ stretch(κ', x) $\setminus \{y\}$;
2. If $\kappa' = \kappa|_{y_1}^{x_1}$ and $\kappa'' = \kappa|_{y_2}^{x_2}$, for $y_1 \neq y_2$, then $I_{\kappa'} \subseteq I_k \setminus$ stretch(κ', y_2);
3. If $\kappa' = \kappa|_{y_1}^{x_1}$ and $\kappa'' = \kappa|_{y_2}^{x_2}$, for $y_1 \neq y_2$, then $I_{\kappa'} \cap I_{k''} = \phi$.

Lemma 3.10. Let κ be in an addition sequence S for a backbone k-tree. Then I_k consists of at most $k+2$ non-empty stretches, and at most $k+1$ if κ is not the base clique of S.

Proof: For the base $(k+1)$-clique $\kappa_0 = \{x_1, x_2, \ldots, x_{k+1}\}$, $x_1 < x_2 < ... < x_{k+1}$, $I_{k_0} = U_{\kappa_0} \backslash \kappa_0$, it is clear that there are at most $(k+2)$ disjoint non-empty sets of consecutive vertices due to Proposition 3.5 (1). Let $\kappa_i = \kappa_0|_{y_i}^{x_i}$ be the children cliques of κ_0. By proposition 3.9,

$I_\kappa = I_{\kappa_0} \setminus \text{stretch}(\kappa_0,\, x_i) \setminus \{y_i\}$. The removal of stretch(κ_0, x_i) reduces the number of disjoint sets to k. Now we consider the effect of including y_i to I_κ, y_i must fall in between x_{j-1} and x_j for some $j \neq i$. y_i may split the set of consecutive vertices stretch(κ_0, y_i) in I_{κ_0} into two, resulting in at most $(k+1)$ disjoints set in I_κ. So in every child κ_i the non-empty stretches are at most $(k+1)$. Assume $\kappa_m \in U_{\kappa_i}$ for some i, $\kappa_i \neq \kappa_m$, has at most $(k+1)$ disjoint set in I_{κ_m}. Now consider any child clique κ_j of κ_m, $\kappa_j = \kappa_m|_{y_j}^{x_j}$, for some j. Similarly the removal of stretch(κ_m, x_j) reduces the disjoint set to at most k and the inclusion of y_j in I_{κ_j} increases the disjoint set to at most $(k+1)$.

Theorem 3.11 Let κ' and κ'' be children of κ in the some k-tree topology for a backbone k-tree, suppose that $\kappa' = \kappa|_{y_i}^{x_i}$ and $\kappa'' = \kappa|_{y_j}^{x_j}$,where y_i and y_j are in the same κ-stretch. Then $\kappa' = \kappa''$.

Proof: We have $y_i \in U_{\kappa'}$ and $y_j \in U_{\kappa''}$ directly. Since y_i and y_j are in the same k-stretch, then also $y_j \in U_{\kappa'}$ by Corallary 3.7.1. Thus y_j must belong to every $(k+1)$-clique on the path from κ' to κ'' in the given tree topology. If $\kappa' \neq \kappa''$ this path must include κ by Proposition 3.3.3. Since $y_j \notin \kappa$ we must have $\kappa' = \kappa''$.

Theorem 3.12. Let S be in an addition sequence for a backbone k-tree. In the rooted tree structure of S the base clique has at most $(k+2)$ children and all other cliques in S have at most $(k+1)$ children.

Proof: By Theorem 3.11, κ has at most one child per stretch, so the bounds follow from Lemma 3.10.

Chapter 4

Algorithms for MskT and MskT-C

Definition 4.0.1. Given any k-clique $K = \{x_1, x_2, \ldots, x_k\}$ in a spanning k-tree on n vertices with backbone B, the backbone B can be decomposed at most into $B_1, \ldots, B_{k+1}$ smaller backbones, where B_i is backbone among the consecutive vertices between x_{i-1} and x_i including x_{i-1} and x_i. For convention $x_0 = 1$ and $x_{k+1} = n$. Define L_i to be $V(B_i)\backslash\{x_{i-1}, x_i\}$.

Lemma 4.0.2. Let $K = \{x_1, x_2, \ldots, x_k\}$; then by proposition 3.4 if $u \in L_i$ then

$$L_i = stretch(K, u).$$

By Lemma 3.10 for a k-clique $K = \{x_1, x_2, \ldots, x_k\}$, with $x_1 < x_2 < \cdots < x_n$ the importable sets I_K consist of at most $k+1$ disjoint non-empty sets delimited by the k vertices. We represent importable sets with vectors $\{0, 1\}^{k+1}$. For K the corresponding importable set $\{l_1, l_1, \ldots, l_{k+1}\}$, $l_i = 0$ and $l_i = 1$ respectively represents the exclusion and inclusion of the vertices in L_i.

The generation of $(k+1)$-clique $\{x_1, x_2, \ldots, x_k, x_p\}$ from k-clique $\{x_1, x_2, \ldots, x_k\}$ does not depend only on the importable set, because we need an indicator r which will help us identify the the first i where a new vertex $x_p \in L_i$ will be introduced among all intervals with $l_i = 1$.

Definition 4.0.3. For given k-clique $K = \{x_1, x_2, \ldots, x_k\}$ and importable set $I\{l_1, l_2, \ldots, l_{k+1}\}$, we let $M_{I\{l_1, l_2, \ldots, l_{k+1}\}}\{x_1, x_2, \ldots, x_k\}$ denote the set of all k-trees on the vertex set $V(K) \cup \bigcup(V(B_i) : l_i = 1)$ with edge set containing $E(K) \cup \bigcup(E(B_i) : l_i = 1)$ and for which K is not a seperating set. This means each of these k-trees is a single branch at K. To avoid overlapping notation the set of trees is defined to be the empty set if $l_i = 0$ and $L_i = \emptyset$ for some i. Then for each i let $M_{I\{l_1, l_2, \ldots, l_{k+1}, r=i\}}\{x_1,\, x_2,\, \ldots,\, x_k\}$ be the set of k-tree topologies for trees in $M_{I\{l_1, l_2, \ldots,\}}\{x_1,\, x_2,\, \ldots,\, x_k\}$ which have root clique equal to $\{x_1,\, x_2,\, \ldots,\, x_{i-1},\, x_p,\, x_i,\, \ldots,\, x_k\}$ for some $x_p \in L_i$. If $l_i = 0$ there is no such x_p, so the specified set of topologies are empty.

Definition 4.0.4.

$$M_{I\{l_1, l_2, \ldots, l_{k+1}\}}\{x_1,\, x_2,\, \ldots,\, x_k\} \equiv_{def} \bigcup_{1 \le i \le k+1} M_{I\{l_1, l_2, \ldots, l_{k+1}, r=i\}}\{x_1,\, x_2,\, \ldots,\, x_k\}.$$

Definition 4.0.5.

$$M'_{I\{l_1, l_2, \ldots, l_{k+1}, r=i\}}\{x_1,\, x_2,\, \ldots,\, x_k\} \equiv_{def} max(\{w(T) : T \in M_{I\{, l_2, \ldots, l_{k+1}, r=i\}}\{x_1,\, x_2,\, \ldots,\, x_k\}\}).$$

Here $w(T)$ is the weight of T for which we are maximizing.

4.1 2-trees

Lemma 4.1.1: A spanning 2-tree $T = (V,\, E)$ with backbone can not have K^4 as a minor.

Lemma 4.1.2. Let T be a spanning 2-tree with backbone B. If T contains the 3-cliques $\{x_i,\, x_j,\, x_p\}$ and $\{x_i,\, x_j,\, x_q\}$ then x_p and x_q must lie in different components of $B - \{x_i,\, x_j\}$.

Proof: We begin with 5 edges in $\{x_i,\, x_j,\, x_p\}$ and $\{x_i,\, x_j,\, x_q\}$. Assume for the contradiction that x_p and x_q lie in the same component of $B\backslash\{x_i, x_j\}$. Then there exists a path from x_p to x_q on that component which is vertex disjoint from $\{x_i, x_j\}$, giving the sixth edge (after contraction) for the K^4 minor. This is contrary to the Lemma 4.1.1.

Note For a 2-tree, choice of a root clique determines a unique tree topology on its 3-cliques. Therefore, in this section of the dissertation the topology is assumed when discussing a 2-tree with a designated root clique. Also in this section a 2-clique is usually referred to as an edge and a 3-clique as a triangle.

Definition 4.1.3 We find it convenient to introduce special notation to describe spanning 2-trees with backbone which are branches at a given root edge $\{x_i, x_j\}$.

1. Let $P(x_i, x_j) = M_{I\{1,0,0\}}\{x_i, x_j\}$ and let $P\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_k, x_i, x_j\}$ for some $k < i$.

2. Let $S(x_i, x_j) = M_{I\{0,0,1\}}\{x_i, x_j\}$ and let $S\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_j, x_k\}$ for some $j < k$.

3. Let $I(x_i, x_j) = M_{I\{0,1,0\}}\{x_i, x_j\}$ and let $I\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_k, x_j\}$ for some $i < k < j$.

4. Let $L(x_i, x_j) = M_{I\{1,1,0\}}\{x_i, x_j\}$ and let $L\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_k, x_j\}$ for some $i < k < j$.

5. Let $R(x_i, x_j) = M_{I\{0,1,1\}}\{x_i, x_j\}$ and let $R\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_k, x_j\}$ for some $i < k < j$.

6. Let $U(x_i, x_j) = M_{I\{1,0,1\}}\{x_i, x_j\}$ and let $U\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_k, x_j\}$ for some $k < i$ or $k > j$.

7. Let $A(x_i, x_j) = M_{I\{1,1,1\}}\{x_i, x_j\}$ and let $A\{x_i, x_k, x_j\}$ denote the members with root clique $\{x_i, x_k, x_j\}$ for some $i < k < j$.

Note that $M_{I(0,0,0)}\{x_i,\, x_j\}$ is just the edge $\{x_i,\, x_j\}$ itself.

Lemma 4.1.4. The four following classes are always empty: $M_{I\{1,1,1,r=1\}}\{x_i,\, x_j\}$, $M_{I\{1,1,1,r=3\}}\{x_i,\, x_j\}$, $M_{I\{1,1,0,r=1\}}\{x_i,\, x_j\}$, and $M_{I\{0,1,1,r=3\}}\{x_i,\, x_j\}$.

Proof. Suppose for example that Z is a branch in the class $M_{I\{1,1,0,r=1\}}\{x_i,\, x_j\}$. By definition the vertices of Z consist of the disjoint sets $L_1, \{x_i\}$, L_2 and $\{x_j\}$ where L_1 and L_2 are non-empty backbone segments. Since Z is is a single branch $\{x_i,\, x_j\}$ there must be some edge joining L_1 directly to L_2; Let $\{x_m,\, x_p\}$ be such an edge, where $x_m \in L_1$ and $x_p \in L_2$. Thus by contracting L_1 we obtain a K^4 minor in Z as shown in the figure 4.1. This is contradictory so no such Z exists. Similar arguements hold for the other three classes.

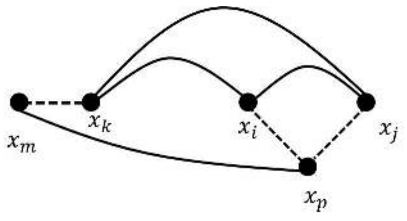


Figure 4.1: 2-tree

Lemma 4.1.5. Let a branch at edge e an e-branch. Then the classes $P(e)$, $S(e)$, $I(e)$, $L(e)$, $R(e)$, $U(e)$ and $A(e)$ partition the set of all e-branches of 2-trees with backbone when $n >= 3$.

Proof It's obvious from Definition 4.1.3, Lemma 4.1.4 that the classes correspond to different importable set sequences (so the classes are disjoint), and they represent all possible importable set sequences except for $I\{0, 0, 0\}$, which is empty for $n >= 3$.

Note that if 2-clique $\{x_i,\, x_j\}$ does not give rise to 3-component then the triangle types will be less than mentioned in Lemma 4.1.5.

Lemma 4.1.6. Let T be a spanning 2-tree with backbone having two or more e-branches. Then the branches must fall in to one of the following cases:

1. $I(e)$, $P(e)$ and $S(e)$;
2. $L(e)$ and $S(e)$;
3. $R(e)$ and $P(e)$;
4. $U(e)$ and $I(e)$.

Proof: These are the only patterns of classes which satisfy the restrictions that distinct e-branches can't share any non-empty stretch (Proposition 3.9.3).

Note that a 2-tree consisting of a single e-branch must belong to one of the seven classes of Definition 4.1.3 when $n \geq 3$.

We now consider how each e-branch of the seven types can be decomposed at its *base triangle,* that is, at the unique 3-clique containing e.

Decomposition of e-branches

For a branch Z of a backbone 2-tree rooted at edge $e = \{x_i, x_j\}$ with base triangle $\{x_i, x_k, x_j\}$ we have seven cases. In stating the results we use the convention that $I'(x_i, x_k) = I(x_i, x_k) \cup \{\{x_i, x_k\}\}$, and so on for the other six classes of branches.

1. **Claim:** If $Z \in I\{x_i, x_k, x_j\}$ then $Z = Z' \cup Z'' \cup \{\{x_i, x_j\}\}$ where $Z' \in I'(x_i, x_k)$ and $Z'' \in I'(x_k, x_j)$.

   Proof: Let Z' be the branch of Z at $\{x_i, x_k\}$ which does not contain x_j, if any, and let Z' be the edge $\{x_i, x_k\}$ if there is no such branch. Likewise Z'' will be $\{x_k, x_j\}$ if Z is a single branch at $\{x_k, x_j\}$, and otherwise is the branch of Z at $\{x_k, x_j\}$ not containing x_i. Note that the backbone $[x_i, x_j]$ of Z becomes $[x_i, x_k] \cup [x_k, x_j]$ where x_k is introduced. Thus if Y is a branch of Z at $\{x_i, x_k\}$ it must contain of all (x_i, x_k) or all of $(x_k, x_j]$, but not both if $(x_i, x_k) \neq \emptyset$. If $(x_i, x_k) = \emptyset$ then $Z' = [x_i, x_k]$. Otherwise $Z' \in I(x_i, x_k)$ since its vertex set is $[x_i, x_k]$. Similarly $Z'' = [x_k, x_j]$ or else has vertex set in $[x_k, x_j]$ and belongs to the class $I(x_k, x_j)$. Now it is easily seen that $E(Z)$ is the disjoint union of $\{\{x_i, x_j\}\}$, $E(Z')$, and $E(Z'')$.

Each of the remaining six cases can be proved by similar arguements concerning the posibilities for branches of Z at $\{x_i, x_k\}$ and $\{x_k, x_j\}$. Below we simply state the results.

2. **Claim:** If $Z \in P\{x_i, x_k, x_j\}$ then (Figure 4.2)

   a) $Z = Z' \cup \{\{x_i, x_j\}, \{x_j, x_k\}\}$ for $Z' \in L'(x_k, x_i)$,

   b) $Z = Z' \cup Z'' \cup \{\{x_i, x_j\}, \{x_j, x_k\}\}$ for $Z' \in I'(x_k, x_i)$ and $Z'' \in P'(x_k, x_i)$, or

   c) $Z = Z' \cup Z'' \cup \{\{x_i, x_j\}\}$ for $Z' \in I'(x_k, x_i)$ and $Z'' \in P'(x_k, x_j)$.

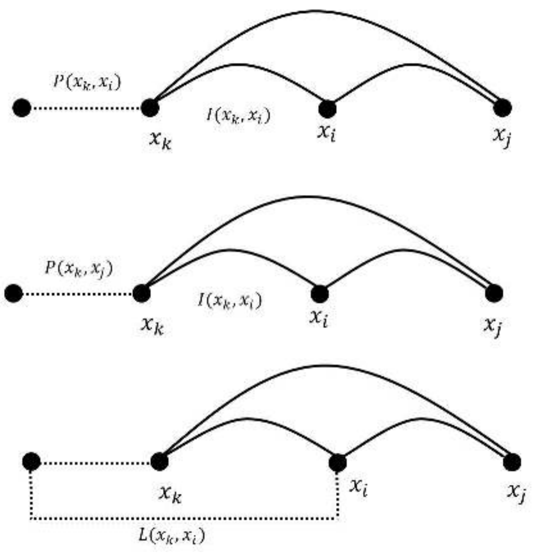


Figure 4.2: $P\{x_i, x_k, x_k\}$

3. **Claim:** If $Z \in S\{x_i, x_k, x_j\}$ then

   a) $Z = Z' \cup \{\{x_i, x_j\}, \{x_i, x_k\}\}$ for $Z' \in R'(x_j, x_k)$,

   b) $Z = Z' \cup Z'' \cup \{\{x_i, x_j\}, \{x_i, x_k\}\}$ for $Z' \in I'(x_j, x_k)$ and $Z'' \in S'(x_j, x_k)$, or

   c) $Z = Z' \cup Z'' \cup \{\{x_i, x_j\}\}$ for $Z' \in I'(x_j, x_k)$ and $Z'' \in S'(x_i, x_k)$.

4. **Claim:** If $Z{\in}L\{x_i,\, x_k,\, x_j\}$ then (Figure 4.3)

   a) $Z = Z' \cup Z''$ for $Z' \in I(x_i,\, x_k,\, x_j)$ and $Z'' \in P'(x_i,\, x_j)$, or

   b) $Z = Z' \cup Z'' \cup \{\{x_i,\, x_j\}\}$ for $Z' \in I'(x_k,\, x_j)$ and $Z'' \in L'(x_i,\, x_k)$.

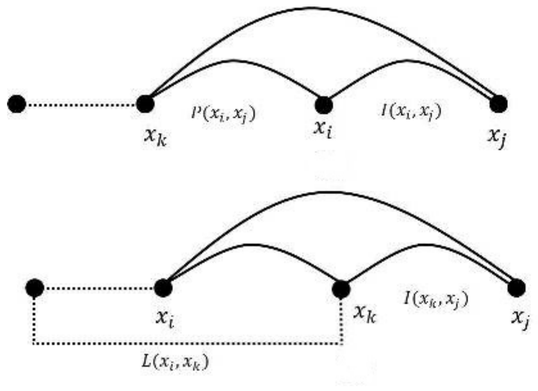


Figure 4.3: $L\{x_i,\, x_k,\, x_k\}$

5. **Claim:** If $Z{\in}R\{x_i,\, x_k,\, x_j\}$ then

   a) $Z = Z' \cup Z''$ for $Z' \in I(x_i,\, x_k,\, x_j)$ and $Z'' \in S'(x_i,\, x_j)$, or

   b) $Z = Z' \cup Z'' \cup \{\{x_i,\, x_j\}\}$ for $Z' \in I'(x_i,\, x_k)$ and $Z'' \in R'(x_k,\, x_j)$.

6. **Claim:** If $Z{\in}U\{x_i,\, x_k,\, x_j\}$ then (Figure 4.4)

   a) $Z = Z' \cup Z'' \cup \{\{x_i,\, x_j\}\}$ for $Z' \in I'(x_j,\, x_k)$ and $Z'' \in U'(x_i,\, x_k)$, or

   b) $Z = Z' \cup Z'' \cup \{\{x_i,\, x_j\}\}$ for $Z' \in I'(x_k,\, x_i)$ and $Z'' \in U'(x_k,\, x_j)$.

7. **Claim:** If $Z{\in}A\{x_i,\, x_k,\, x_j\}$ then $Z = Z' \cup Z'' \cup \{\{x_i,\, x_j\}\}$ for $Z' \in L'(x_i,\, x_k)$ and $Z'' \in R'(x_k,\, x_j)$.

The cliques which may be wighted in a 2-tree T will either be edges or triangles. Then the objective function for T may be

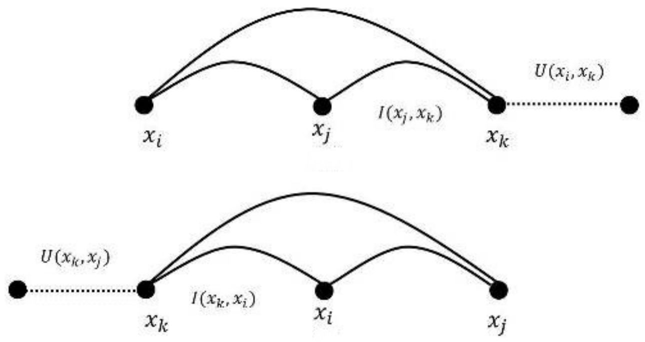


Figure 4.4: $U\{x_i,\, x_k,\, x_k\}$

1.
$$\hat{w}(T) = \sum_{e \in E(T)} w(e), \qquad or$$

2.
$$\hat{w}(T) = \sum_{\alpha \in \Delta(T)} w(\alpha)$$

where $\Delta(T)$ is the set of triangles (3-cliques) in T.

Let $\hat{I}(x_i,\, x_j) = max\{w(T) : T \in I(x_i,\, x_j)\}$, and so on for the other six classes of branches at $\{x_i,\, x_j\}$. For a weight function of type 1 we obtain recursive equations for $\hat{I}(x_i,\, x_j)$, *etc.*, from the decomposition claims in the obvious way, making use of the independence of choices for the sub branches Z' and Z'' in each case.

$$\hat{I}(x_i,\, x_j) = max_{i<k<j} \begin{cases} w\{x_i,\, x_j\} & if \quad i+1=j, \\ w\{x_i,\, x_j\} + \hat{I}(x_i,\, x_k) + \hat{I}(x_k,\, x_j) & otherwise. \end{cases}$$

$$\hat{P}(x_i, x_j) = max_{k<i} \begin{cases} 0 & if \quad i = 1,\ or\ else \\ w\{x_i,\ x_j\} + max \begin{cases} \hat{I}(x_k,\ x_i) + \hat{P}(x_k,\ x_i), \\ \hat{I}(x_k,\ x_i) + \hat{P}(x_k,\ x_j), \\ \hat{L}(x_k,\ x_i). \end{cases} \end{cases}$$

$$\hat{S}(x_i, x_j) = max_{j<k} \begin{cases} 0 & if \quad j = n,\ or\ else \\ w\{x_i,\ x_j\} + max \begin{cases} \hat{I}(x_j,\ x_k) + \hat{S}(x_i,\ x_k), \\ \hat{R}(x_j,\ x_k), \\ \hat{I}(x_j,\ x_k) + \hat{S}(x_j,\ x_k). \end{cases} \end{cases}$$

$$\hat{L}(x_i, x_j) = \begin{cases} \hat{I}(x_i,\ x_j) & if \quad i = 1,\ or\ else \\ max \begin{cases} \hat{I}(x_i,\ x_j) + \hat{P}(x_i,\ x_j), \\ max_{i<k<j} \left\{ w\{x_i,\ x_j\} + \hat{I}(x_k,\ x_j) + \ \hat{L}(x_i,\ x_k) \right\}. \end{cases} \end{cases}$$

$$\hat{R}(x_i, x_j) = \begin{cases} \hat{I}(x_i,\ x_j) & if \quad j = n,\ or\ else \\ max \begin{cases} \hat{I}(x_i,\ x_j) + \hat{S}(x_i,\ x_j), \\ max_{i<k<j}\{w\{x_i,\ x_j\} + \hat{I}(x_i,\ x_k) + \ \hat{R}(x_k,\ x_j)\}. \end{cases} \end{cases}$$

$$\hat{U}(x_i, x_j) = w\{x_i,\, x_j\} + \begin{cases} max_{k>j} \begin{cases} max \begin{cases} \hat{P}(x_j,\, x_n) \quad +\hat{I}(x_j,\, n) \\ \hat{P}(x_i,\, x_j) \quad +\hat{I}(x_j,\, n) \end{cases} & if \quad k = n,\ or\ else \\ \hat{I}(x_j,\, x_k) + \hat{U}(x_i,\, x_k). \end{cases} \\ max_{k<i} \begin{cases} max \begin{cases} \hat{S}(x_1,\, x_j) + \quad \hat{I}(1,\, x_i) \\ \hat{P}(x_i,\, x_j) + \quad \hat{I}(1,\, x_i) \end{cases} & if \quad k = 1,\ or\ else \\ \hat{I}(x_k,\, x_i) + \hat{U}(x_k,\, x_j). & else \end{cases} \end{cases}$$

$$\hat{A}(x_i,\, x_j) = max\{w\{x_i,\, x_j\} + \hat{L}(x_i,\, x_k) + \hat{R}(x_k,\, x_j) :\ i < k < j\}.$$

Similarly for a weight function of type 2 we obtain recursive equations for $\hat{I}(x_i,\, x_j)$, *etc.*

$$\hat{I}(x_i,\, x_j) = max_{i<k<j} \begin{cases} 0 & if \quad i + 1 = j, \\ w\{x_i,\, x_k,\, x_j\} + \hat{I}(x_i,\, x_k) + \hat{I}(x_k,\, x_j) & otherwise \end{cases}$$

$$\hat{P}(x_i, x_j) = max_{k<i} \begin{cases} 0 & if \quad i = 1,\ or\ else \\ w\{x_k,\, x_i,\, x_j\} + max \begin{cases} \hat{I}(x_k,\, x_j) + \hat{P}(x_k,\, x_i), \\ \hat{I}(x_k,\, x_i) + \hat{P}(x_k,\, x_j), \\ \hat{L}(x_k,\, x_i). \end{cases} \end{cases}$$

$$\hat{S}(x_i, x_j) = max_{j<k} \begin{cases} 0 & if \quad j = n, \; or\, else \\ w\{x_i,\, x_j,\, x_k\} + max \begin{cases} \hat{I}(x_j,\, x_k) + \hat{S}(x_i,\, x_k), \\ \hat{R}(x_j,\, x_k), \\ \hat{I}(x_j,\, x_k) + \hat{S}(x_j,\, x_k). \end{cases} \end{cases}$$

$$\hat{L}(x_i, x_j) = \begin{cases} \hat{I}(x_i,\, x_j) & if \quad i = 1, \; or\, else \\ max \begin{cases} \hat{I}(x_i,\, x_j) + \hat{P}(x_i,\, x_j), \\ max_{i<k<j} \begin{cases} w\{x_i,\, x_k,\, x_j\} + \hat{I}(x_k,\, x_j) + \; \hat{L}(x_i,\, x_k)\}. \end{cases} \end{cases} \end{cases}$$

$$\hat{R}(x_i, x_j) = \begin{cases} \hat{I}(x_i,\, x_j) & if \quad j = n, \; or\, else \\ max \begin{cases} \hat{I}(x_i,\, x_j) + \hat{S}(x_i,\, x_j), \\ max_{i<k<j}\{w\{x_i,\, x_k,\, x_j\} + \hat{I}(x_i,\, x_k) + \; \hat{R}(x_k,\, x_j)\}. \end{cases} & else \end{cases}$$

$$\hat{U}(x_i, x_j) = w\{x_i,\, x_j,\, x_k\} + \begin{cases} max_{k>j} \begin{cases} max \begin{cases} \hat{P}(x_j,\, x_n) \quad +\hat{I}(x_j,\, n) \\ \hat{P}(x_i,\, x_j) \quad +\hat{I}(x_j,\, n) \end{cases} & if \quad k = n, \; or\, else \\ \hat{I}(x_j,\, x_k) + \hat{U}(x_i,\, x_k). \end{cases} \\ max_{k<i} \begin{cases} max \begin{cases} \hat{S}(x_1,\, x_j) + \quad \hat{I}(1,\, x_i) \\ \hat{P}(x_i,\, x_j) + \quad \hat{I}(1,\, x_i) \end{cases} & k = 1 \\ \hat{I}(x_k,\, x_i) + \hat{U}(x_k,\, x_j). \end{cases} \end{cases}$$

$$\hat{A}(x_i, x_j) = max\{w\{x_i, x_k, x_j\} + \hat{L}(x_i, x_k) + \hat{R}(x_k, x_j) : i < k < j\}.$$

Theorem 4.1.7 The maximization problems Ms2T and Ms2T-C can be solved for a given weighted complete graph on vertex set $\{1, 2, \cdots, n\}$ in time $O(n^3)$ and space $O(n^2)$.

Proof: The answer is $\hat{S}(1, 2)$, where the maximization is with respect to a weight function of type 1 for Ms2T and type 2 for Ms2T-C. In either case the recurrences above enable computation of the values $\hat{I}(i, j)$, *etc.*, for $1 \leq i < j \leq n$. By storing these $O(n^2)$ values progressively, each new value can be computed in $O(n)$ time in terms of already stored values. For $\hat{I}(i, j)$ the recursion is based on the difference $d = j - i$ starting with $d = 1$. For $\hat{S}(i, j)$ the recursion will be on $d = n - i$, *etc.*

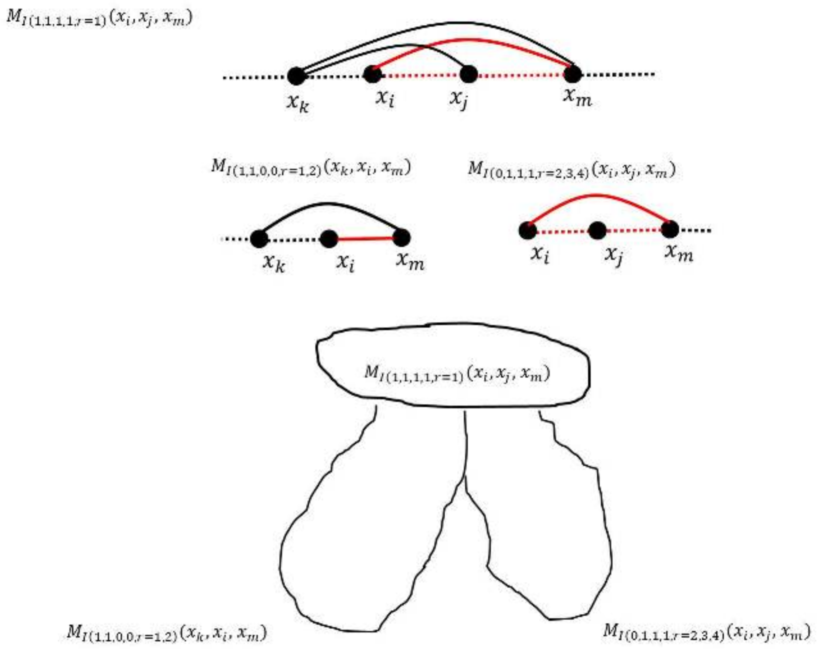
The computation of $\hat{I}(i, j)$, *etc.*, can be modified in the usual way by memoization to allow for the recovery of a k-tree with maximum weight, again in time $O(n^3)$ and space $O(n^2)$.

4.2 Finding Optimal k-trees for k=3

In this section we will look through the example of finding optimal spanning k-tree, $k = 3$, in detail.

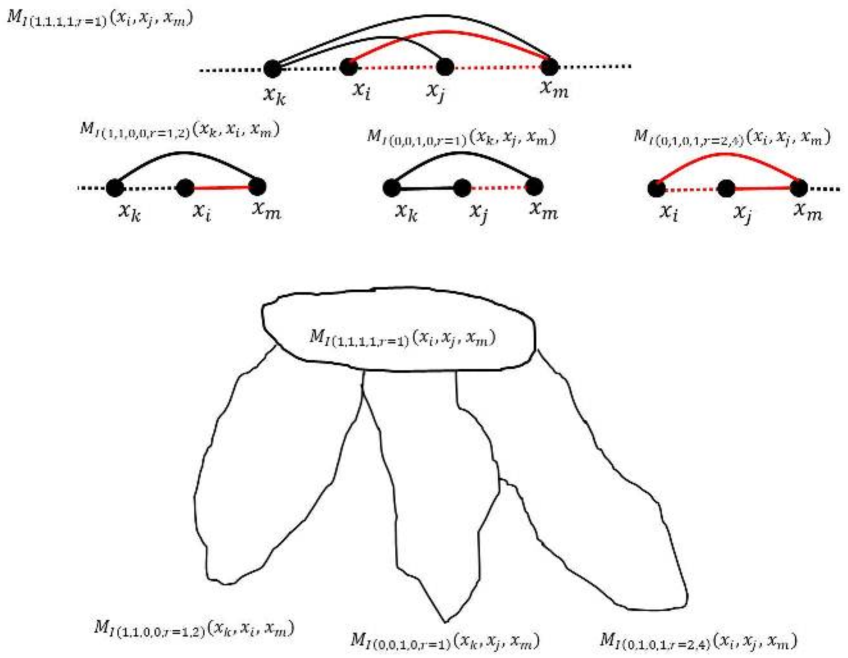
Assume spanning 3-tree T is grown from a 3-clique $\{x_i, x_j, x_m\}$, $i < j < m$. The introduction of a new vertex x_k in any open region will give rise to a 4-clique. This 4-clique has 4 3-cliques as shown in the figure 4.10 with 5 open regions. We can express this 4-clique as a combination of 4 3-cliques such that the 5 open regions will be distributed into these 4 3-cliques. Each clique can be visualized as a child of the $(k + 1)$-clique and at most there will be $(k + 1)$ children. Recursively each 3-clique represent a spanning 3-tree rooted at this clique with open regions.

If the spanning 3-tree rooted at $\{x_k,\ x_i,\ x_j,\ x_m\}$ has two, three or four children then the 5 open regions will be distributed among two, three or four 3-cliques correspondingly as shown in figure 4.5, 4.6, 4.7.



$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
x_k
x_i
x_j
x_m
$M_{I(1,1,0,0,r=1,2)}(x_k, x_i, x_m)$
$M_{I(0,1,1,1,r=2,3,4)}(x_i, x_j, x_m)$
$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
$M_{I(1,1,0,0,r=1,2)}(x_k, x_i, x_m)$
$M_{I(0,1,1,1,r=2,3,4)}(x_i, x_j, x_m)$

Figure 4.5: 2-children


$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
x_k
x_i
x_j
x_m
$M_{I(1,1,0,0,r=1,2)}(x_k, x_i, x_m)$
$M_{I(0,0,1,0,r=1)}(x_k, x_j, x_m)$
$M_{I(0,1,0,1,r=2,4)}(x_i, x_j, x_m)$
$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
$M_{I(1,1,0,0,r=1,2)}(x_k, x_i, x_m)$
$M_{I(0,0,1,0,r=1)}(x_k, x_j, x_m)$
$M_{I(0,1,0,1,r=2,4)}(x_i, x_j, x_m)$

□

Figure 4.6: 3-children

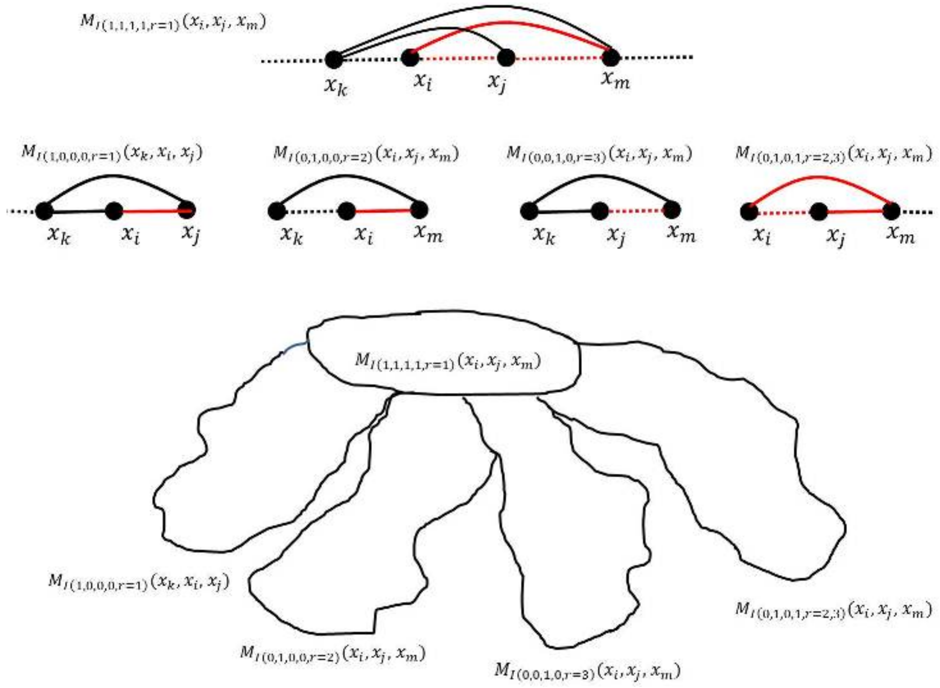

$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
x_k
x_i
x_j
x_m
$M_{I(1,0,0,0,r=1)}(x_k, x_i, x_j)$
$M_{I(0,1,0,0,r=2)}(x_i, x_j, x_m)$
$M_{I(0,0,1,0,r=3)}(x_i, x_j, x_m)$
$M_{I(0,1,0,1,r=2,3)}(x_i, x_j, x_m)$
$M_{I(1,1,1,1,r=1)}(x_i, x_j, x_m)$
$M_{I(1,0,0,0,r=1)}(x_k, x_i, x_j)$
$M_{I(0,1,0,0,r=2)}(x_i, x_j, x_m)$
$M_{I(0,0,1,0,r=3)}(x_i, x_j, x_m)$
$M_{I(0,1,0,1,r=2,3)}(x_i, x_j, x_m)$

Figure 4.7: 4-branches

Recursive equations for the fixed 3-clique $\{x_i,\, x_j,\, x_m\}$

The number of recursive equations which we need to consider for the fixed clique 3-clique $\{x_i,\, x_j,\, x_m\}$ is $\binom{4}{1} + \binom{4}{2} \times 2 + \binom{4}{3} \times 3 + 4 = 32$. We will only document the representative recursive equations for this fixed 3-clique. We will sometime use the encoding $\{x_i,\, x_j,\, x_m\} = \{i,\, j,\, m\}$ and $I\{1, 1, 1, 0, r = 1, 2, 3\} = \{1110, 123\}$ whenever necessary.

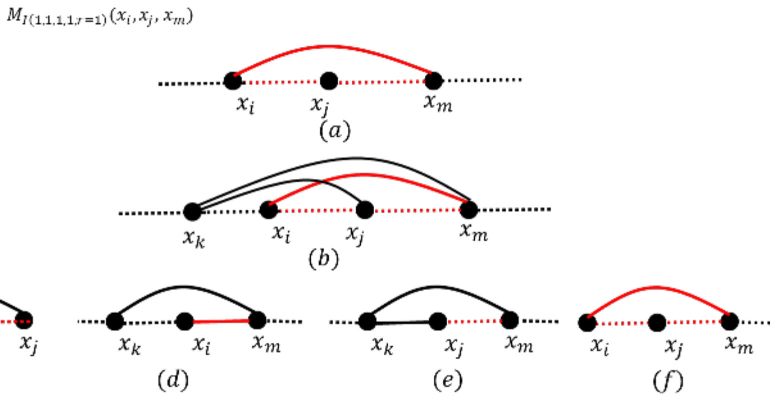


Figure 4.8: $\mathbf{M_{(1,1,1,1,r=1)}}(x_i, x_j, x_m)$

Recursive equation for $\mathbf{M'_{\{1,0,0,0,r=1\}}}\{i,j,m\}$ (see Figure 4.8).

$$M'_{I\{1,1,1,1,r=1\}} = Max_{k<i}\Big\{f(k,i,j,m)+$$

$$\begin{cases}
M'_{I\{1,1,1,0,r=1,2,3\}}\{k,\, i,\, j\} + M'_{I\{0,0,1,1,r=3,4\}}\{k,\, j,\, m\} \\
M'_{I\{0,1,1,0,r=2,3\}}\{k,\, i,\, j\} + M'_{I\{1,0,1,1,r=1,3,4\}}\{k,\, j,\, m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{k,\, i,\, j\} + M'_{I\{0,1,1,1,r=2,3,4\}}\{i,\, j,\, m\} \\
M'_{I\{1,1,1,0,r=1,2,3\}}\{k,\, i,\, j\} + M'_{I\{0,0,1,1,r=3,4\}}\{i,\, j,\, m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{k,\, i,\, m\} + M'_{I\{0,1,1,1,r=2,3,4\}}\{i,\, j,\, m\} \\
M'_{I\{1,1,0,1,r=1,2,4\}}\{k,\, i,\, m\} + M'_{I\{0,1,1,1,r=2,3,4\}}\{i,\, j,\, m\} \\
M'_{I\{1,1,1,0,r=1,2,3\}}\{k,\, i,\, j\} + M'_{I\{0,0,0,1,r=4\}}\{k,\, i,\, m\} + M'_{I\{0,0,1,0,r=3\}}\{k,\, j,\, m\} \\
M'_{I\{0,1,1,0,r=2,3\}}\{k,\, i,\, j\} + M'_{I\{1,0,0,1,r=1,4\}}\{k,\, i,\, m\} + M'_{I\{0,0,1,0,r=3\}}\{k,\, j,\, m\} \\
M'_{I\{1,0,1,0,r=1,3\}}\{k,\, i,\, j\} + M'_{I\{0,1,0,1,r=2,4\}}\{k,\, i,\, m\} + M'_{I\{0,0,1,0,r=3\}}\{k,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{1101,1,2,4\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} \\
M'_{I\{1010,1,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{0011,3,4\}}\{k,\, j,\, m\} \\
M'_{I\{0110,2,3\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0011,3,4\}}\{k,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{1100,1,2\}}\{k,\, i,\, m\} + M'_{I\{0011,3,4\}}\{k,\, j,\, m\} \\
M'_{I\{0110,2,3\}}\{k,\, i,\, j\} + M'_{I\{0001,1\}}\{k,\, i,\, m\} + M'_{I\{1010,1,3\}}\{k,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{1011,1,3,4\}}\{k,\, j,\, m\} \\
M'_{I\{1110,1,2,3\}}\{k,\, i,\, j\} + M'_{I\{0001,4\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\}
\end{cases}$$

$$\begin{cases}
M'_{I\{0110,2,3\}}\{k,\,i,\,j\}+M'_{I\{1001,1,4\}}\{k,\,i,\,m\}+M'_{I\{0010,3\}}\{i,\,j,\,m\}\\
M'_{I\{1010,1,3\}}\{k,\,i,\,j\}+M'_{I\{0101,2,4\}}\{k,\,i,\,m\}+M'_{I\{0010,3\}}\{i,\,j,\,m\}\\
M'_{I\{0010,3\}}\{k,\,i,\,j\}+M'_{I\{1101,1,2,4\}}\{k,\,i,\,m\}+M'_{I\{0010,3\}}\{i,\,j,\,m\}\\
M'_{I\{0110,2,3\}}\{k,\,i,\,j\}+M'_{I\{1000,1\}}\{k,\,i,\,m\}+M'_{I\{0011,3,4\}}\{i,\,j,\,m\}\\
M'_{I\{1010,1,3\}}\{k,\,i,\,j\}+M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{0011,3,4\}}\{i,\,j,\,m\}\\
M'_{I\{0010,3\}}\{k,\,i,\,j\}+M'_{I\{1100,1,2\}}\{k,\,i,\,m\}+M'_{I\{0011,3,4\}}\{i,\,j,\,m\}\\
M'_{I\{1000,1\}}\{k,\,i,\,j\}+M'_{I\{0101,2,4\}}\{k,\,i,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,j\}+M'_{I\{1001,1,4\}}\{k,\,i,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{1100,1,2\}}\{k,\,i,\,j\}+M'_{I\{1101,1,2,4\}}\{k,\,i,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{1000,1\}}\{k,\,i,\,j\}+M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{0111,2,3,4\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,j\}+M'_{I\{1000,1\}}\{k,\,i,\,m\}+M'_{I\{0111,2,3,4\}}\{i,\,j,\,m\}\\
M'_{I\{1101,1,2,4\}}\{k,\,i,\,m\}+M'_{I\{0010,3\}}\{k,\,j,\,m\}+M'_{I\{0100,2\}}\{i,\,j,\,m\}\\
M'_{I\{1100,1,2\}}\{k,\,i,\,m\}+M'_{I\{0011,3,4\}}\{k,\,j,\,m\}+M'_{I\{0100,2\}}\{i,\,j,\,m\}\\
M'_{I\{0101,2,4\}}\{k,\,i,\,m\}+M'_{I\{1010,2,4\}}\{k,\,j,\,m\}+M'_{I\{0100,2\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{1011,1,3,4\}}\{k,\,j,\,m\}+M'_{I\{0100,2\}}\{i,\,j,\,m\}\\
M'_{I\{1100,1,2\}}\{k,\,i,\,m\}+M'_{I\{0010,3\}}\{k,\,j,\,m\}+M'_{I\{0101,2,4\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{1010,1,3\}}\{k,\,j,\,m\}+M'_{I\{0101,2,4\}}\{i,\,j,\,m\}\\
M'_{I\{0101,2,4\}}\{k,\,i,\,m\}+M'_{I\{1000,1\}}\{k,\,j,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{1100,1,2\}}\{k,\,i,\,m\}+M'_{I\{0001,4\}}\{k,\,j,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{1001,1,4\}}\{k,\,j,\,m\}+M'_{I\{0110,2,3\}}\{i,\,j,\,m\}\\
M'_{I\{0100,2\}}\{k,\,i,\,m\}+M'_{I\{1000,1\}}\{k,\,j,\,m\}+M'_{I\{0111,2,3,4\}}\{i,\,j,\,m\}
\end{cases}$$

$$\left\{\begin{array}{l}
M'_{I\{0110,r=2,3\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{0110,2,3\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{1010,1,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{1100,1,2\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{1010,1,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{1010,1,3\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0001,4\}}\{i,\, j,\, m\} \\
M'_{I\{1100,1,2\}}\{k,\, i,\, j\} + M'_{I\{0001,4\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0100,2\}}\{i,\, j,\, m\} \\
M'_{I\{1000,1\}}\{k,\, i,\, j\} + M'_{I\{0101,2,4\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0100,2\}}\{i,\, j,\, m\} \\
M'_{I\{0100,2\}}\{k,\, i,\, j\} + M'_{I\{1001,1,4\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0100,2\}}\{i,\, j,\, m\} \\
M'_{I\{0100,2\}}\{k,\, i,\, j\} + M'_{I\{0001,4\}}\{k,\, i,\, m\} + M'_{I\{1010,1,3\}}\{k,\, j,\, m\} + M'_{I\{0100,2\}}\{i,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{1100,1,2\}}\{k,\, i,\, m\} + M'_{I\{0001,4\}}\{k,\, j,\, m\} + M'_{I\{001,0,3\}}\{i,\, j,\, m\} \\
M'_{I\{1010,1,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{0001,4\}}\{k,\, j,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\} \\
M'_{I\{0110,2,3\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0001,3\}}\{k,\, j,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\} \\
M'_{I\{0110,2,3\}}\{k,\, i,\, j\} + M'_{I\{0001,4\}}\{k,\, i,\, m\} + M'_{I\{1000,1\}}\{k,\, j,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{0101,2,4\}}\{k,\, i,\, m\} + M'_{I\{1000,1\}}\{k,\, j,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\} \\
M'_{I\{0010,3\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{1001,1,4\}}\{k,\, j,\, m\} + M'_{I\{0010,3\}}\{i,\, j,\, m\} \\
M'_{I\{1000,1\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0101,2,4\}}\{i,\, j,\, m\} \\
M'_{I\{0100,2\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0010,3\}}\{k,\, j,\, m\} + M'_{I\{0101,2,4\}}\{i,\, j,\, m\} \\
M'_{I\{1000,1\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{0001,4\}}\{k,\, j,\, m\} + M'_{I\{0110,2,3\}}\{i,\, j,\, m\} \\
M'_{I\{0100,2\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{0001,4\}}\{k,\, j,\, m\} + M'_{I\{0110,2,3\}}\{i,\, j,\, m\} \\
M'_{I\{1000,1\}}\{k,\, i,\, j\} + M'_{I\{0001,4\}}\{k,\, i,\, m\} + M'_{I\{1000,1\}}\{k,\, j,\, m\} + M'_{I\{0110,2,3\}}\{i,\, j,\, m\} \\
M'_{I\{0100,2\}}\{k,\, i,\, j\} + M'_{I\{1000,1\}}\{k,\, i,\, m\} + M'_{I\{1000,1\}}\{k,\, j,\, m\} + M'_{I\{0011,3,4\}}\{i,\, j,\, m\} \\
M'_{I\{1000,1\}}\{k,\, i,\, j\} + M'_{I\{0100,2\}}\{k,\, i,\, m\} + M'_{I\{1000,1\}}\{k,\, j,\, m\} + M'_{I\{0011,3,4\}}\{i,\, j,\, m\}
\end{array}\right.$$

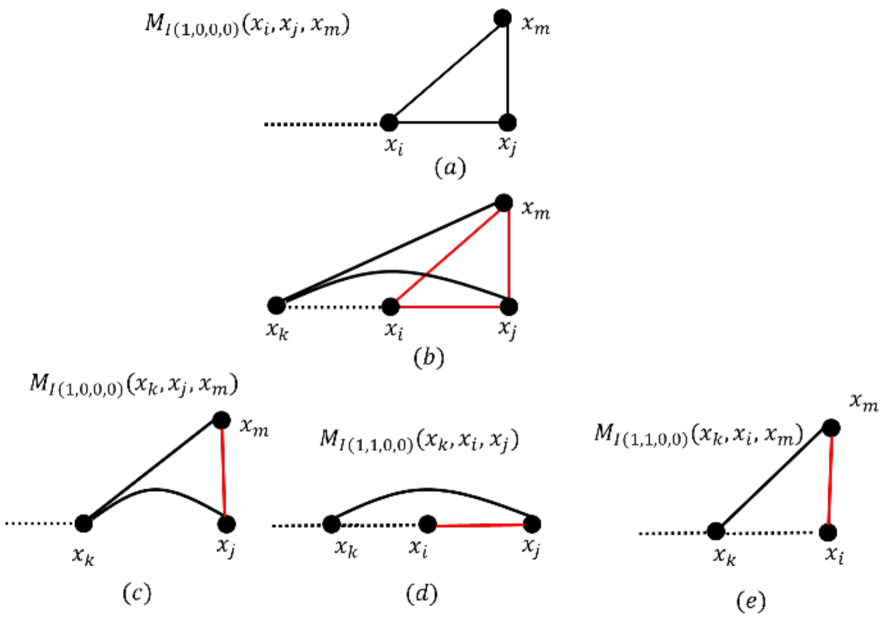

$M_{I(1,0,0,0)}(x_i, x_j, x_m)$
x_m
x_i
x_j
(a)
x_m
x_k
x_i
x_j
(b)
$M_{I(1,0,0,0)}(x_k, x_j, x_m)$
x_m
x_k
x_j
(c)
$M_{I(1,1,0,0)}(x_k, x_i, x_j)$
x_k
x_i
x_j
(d)
$M_{I(1,1,0,0)}(x_k, x_i, x_m)$
x_m
x_k
x_i
(e)

Figure 4.9: $\mathbf{M_{(1,0,0,0,r=1)}}(x_i, x_j, x_m)$

Recursive equation for $\mathbf{M'_{I\{1,0,0,0,r=1\})}}\{x_i, x_j, x_m\}$ (see Figure 4.9).

$$M_{I\{1,0,0,0,r=1\})}\{x_i,\, x_j,\, x_m\} = max_{k<i}\{f(x_k, x_i, x_j, x_m)+$$

$$max\begin{cases} M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} \\ M'_{I\{0,1,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{1,0,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} \\ M'_{I\{0,1,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} & \\ M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} & \\ M'_{I\{0,1,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{1,1,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & \\ M'_{I\{1,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & \end{cases}$$

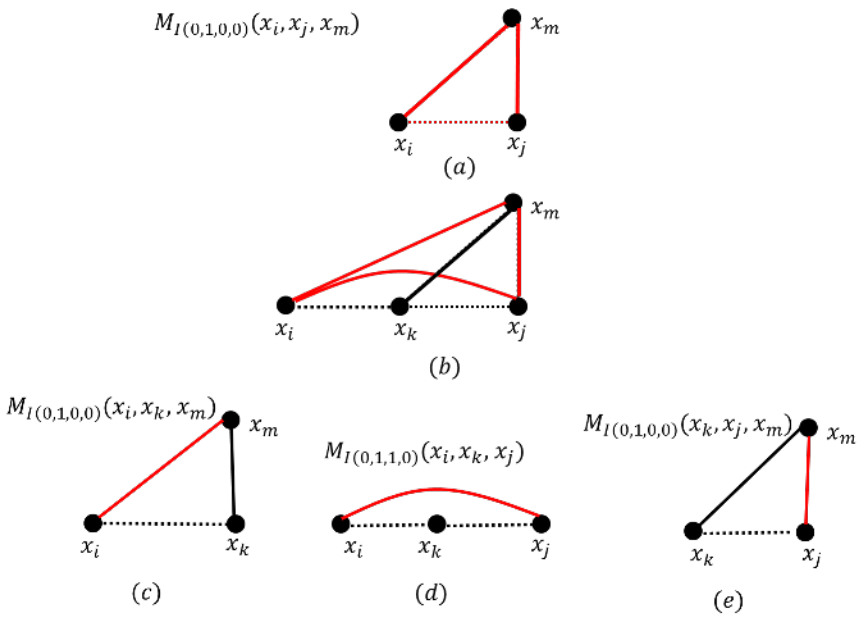
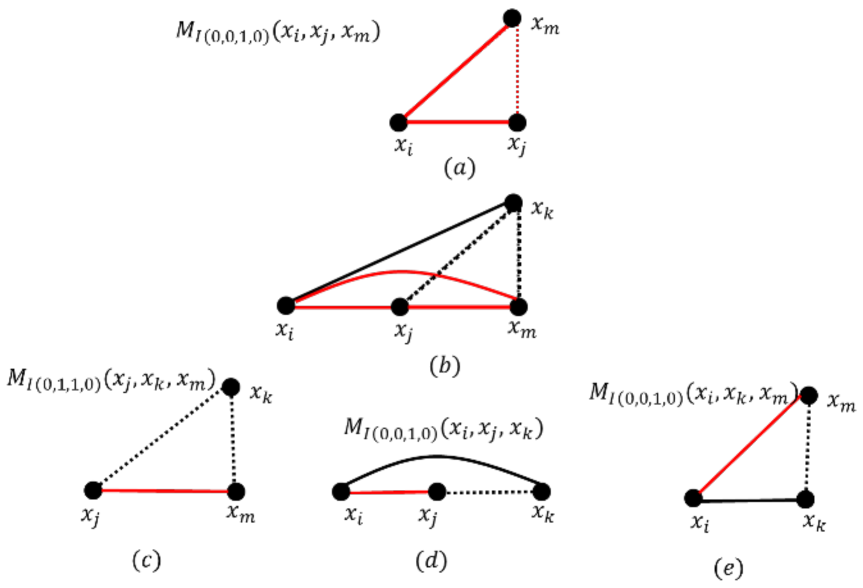

$M_{I(0,1,0,0)}(x_i, x_j, x_m)$
x_m
x_i
x_j
(a)
x_m
x_i
x_k
x_j
(b)
$M_{I(0,1,0,0)}(x_i, x_k, x_m)$
x_m
x_i
x_k
(c)
$M_{I(0,1,1,0)}(x_i, x_k, x_j)$
x_i
x_k
x_j
(d)
$M_{I(0,1,0,0)}(x_k, x_j, x_m)$
x_m
x_k
x_j
(e)

Figure 4.10: $\mathbf{M_{(0,1,0,0,r=2)}}(x_i, x_j, x_m)$

Recursive equation for $\mathbf{M'_{I\{1,0,0,0,r=1\}}}\{x_i, x_j, x_m\}$ (see Figure 4.10).

$$M_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} = max_{i<k<j}\{f(x_i,\, x_k,\, x_j,\, x_m)+$$

$$max\begin{cases} M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} \\ M'_{I\{0,1,1,0,r=2\}}\{x_i,\, x_k,\, x_j\} & \\ M'_{I\{0,1,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \end{cases}$$



$M_{I(0,0,1,0)}(x_i, x_j, x_m)$
x_m
x_i
x_j
(a)
x_k
x_i
x_j
x_m
(b)
$M_{I(0,1,1,0)}(x_j, x_k, x_m)$
x_k
x_j
x_m
(c)
$M_{I(0,0,1,0)}(x_i, x_j, x_k)$
x_i
x_j
x_k
(d)
$M_{I(0,0,1,0)}(x_i, x_k, x_m)$
x_m
x_i
x_k
(e)

Figure 4.11: $\mathbf{M_{(0,0,1,0,r=3)}}(x_i, x_j, x_m)$

Recursive equation for $\mathbf{M'_{I\{1,0,0,0,r=1\}}}\{x_i, x_j, x_m\}$ (see Figure4.11).

$$M_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} = max_{j<k<m}\{f(x_i,\, x_j,\, x_k,\, x_m)+$$

$$max\begin{cases} M'_{I\{0,0,1,0,r=3\}}\{x_j,\, x_k,\, x_m\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_k\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_j,\, x_k,\, x_m\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_m\} \\ M'_{I\{0,1,1,0,r=2\}}\{x_j,\, x_k,\, x_m\} & \\ M'_{I\{0,1,1,0,r=3\}}\{x_j,\, x_k,\, x_m\} & \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{0,0,1,0,r=2\}}\{x_i,\, x_j,\, x_k\} \end{cases}$$

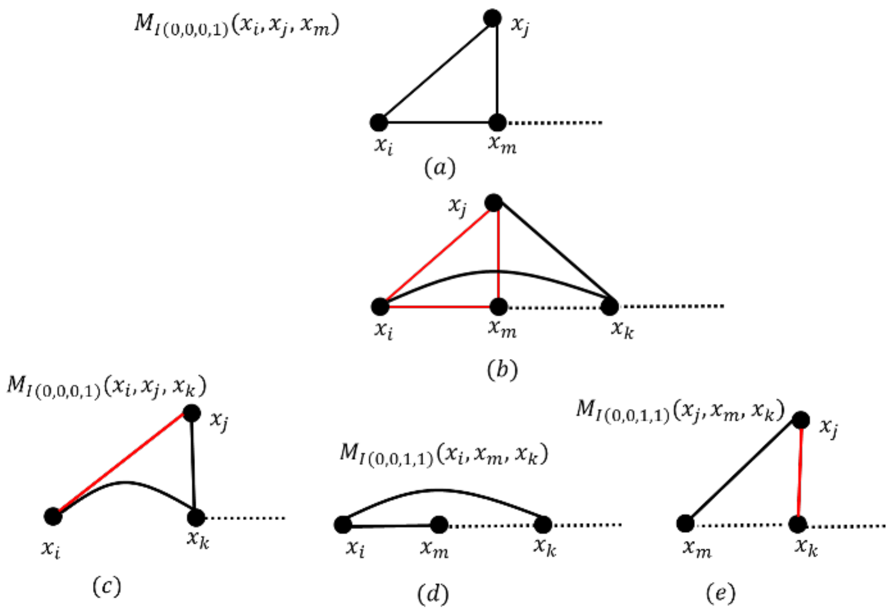

$M_{I(0,0,0,1)}(x_i, x_j, x_m)$
x_j
x_i
x_m
(a)
x_j
x_i
x_m
x_k
(b)
$M_{I(0,0,0,1)}(x_i, x_j, x_k)$
x_j
x_i
x_k
(c)
$M_{I(0,0,1,1)}(x_i, x_m, x_k)$
x_i
x_m
x_k
(d)
$M_{I(0,0,1,1)}(x_j, x_m, x_k)$
x_j
x_m
x_k
(e)

Figure 4.12: $\mathbf{M_{(0,0,0,1,r=4)}}(x_i, x_j, x_m)$

Recursive equation for $\mathbf{M'_{I\{0,0,0,1,r=4\}}}\{x_i, x_j, x_m\}$ (see Figure 4.12).

$$M_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} = max_{m<k}\{f(x_i,\, x_j,\, x_m,\, x_k)+$$

$$max\begin{cases} M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_m,\, x_k\} & +M'_{I\{0,0,1,0,r=3\}}\{x_j,\, x_m,\, x_k\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_m,\, x_k\} & +M'_{I\{0,0,0,1,r=4\}}\{x_j,\, x_m,\, x_k\} \\ M'_{I\{0,0,1,1,r=4\}}\{x_i,\, x_m,\, x_k\} & \\ M'_{I\{0,0,1,1,r=3\}}\{x_i,\, x_m,\, x_k\} & \\ M'_{I\{0,0,1,0,r=3\}}\{x_j,\, x_m,\, x_k\} & +M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_k\} \\ M'_{I\{0,0,0,1,r=4\}}\{x_j,\, x_m,\, x_k\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_k\} \\ M'_{I\{0,0,1,1,r=3\}}\{x_j,\, x_m,\, x_k\} & \\ M'_{I\{0,0,1,1,r=4\}}\{x_j,\, x_m,\, x_k\} & \end{cases}$$

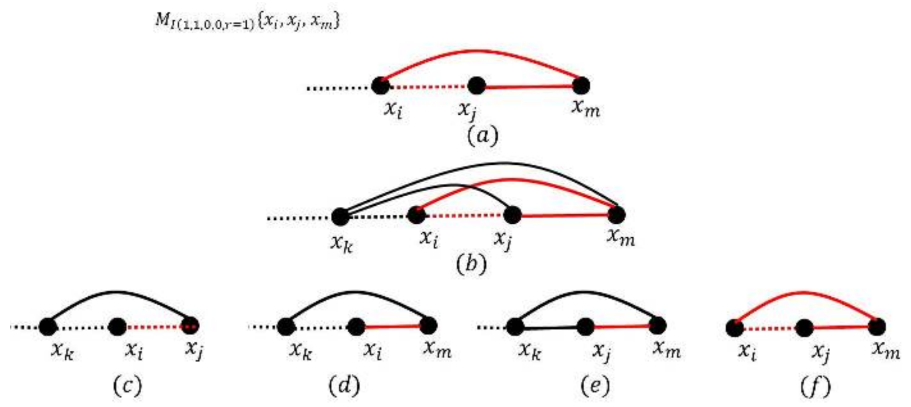


Figure 4.13: $\mathbf{M_{(1,1,0,0,r=1)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{I\{1,1,0,0,r=1\}}}\{x_i, x_j, x_m\}$ (see Figure 4.13).

$$M_{I\{1,1,0,0,r=1\}}\{x_i,\, x_j,\, x_m\} = max_{m<k}\{f(x_k,\, x_i,\, x_j,\, x_m)+$$

$$max\begin{cases} M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} \\ M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} \\ M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{1,1,1,0,r=1,2,3\}}\{x_i,\, x_m,\, x_k\} & \\ M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,1,0,1,r=2\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_k\} \\ M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \end{cases}$$

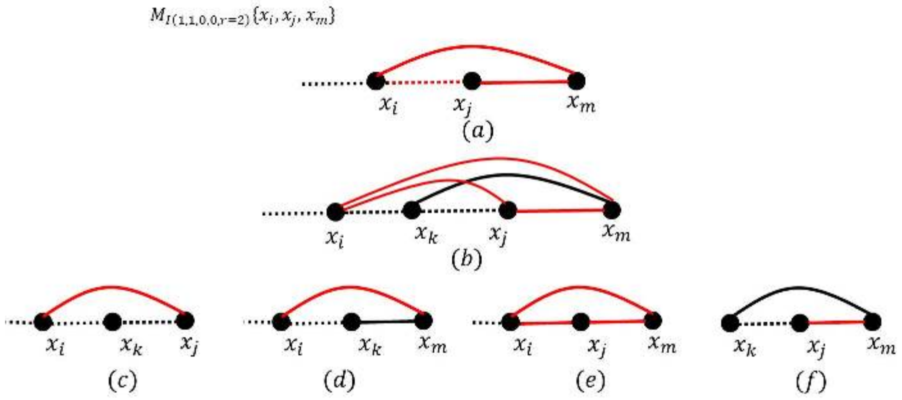


Figure 4.14: $\mathbf{M_{(1,1,0,0,r=2)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{I\{1,1,0,0,r=2\}}}\{x_i, x_j, x_m\}$ (see Figure 4.14).

$$M'_{I\{1,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} = max_{i<k<j}\{f(x_i,\, x_k,\, x_j,\, x_m)+$$

$$max\begin{cases} M'_{I\{1,0,1,0,r=1,3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} \\ M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_k,\, x_m\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{1,1,0,0,r=1,2\}}\{x_i,\, x_k,\, x_m\} \\ M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{1,1,0,0,r=1,2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{1,1,0,0,r=1,2\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_j,\, x_k,\, x_m\} + M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_k,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{1,0,0,0,r=1\}}\{x_i,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \end{cases}$$

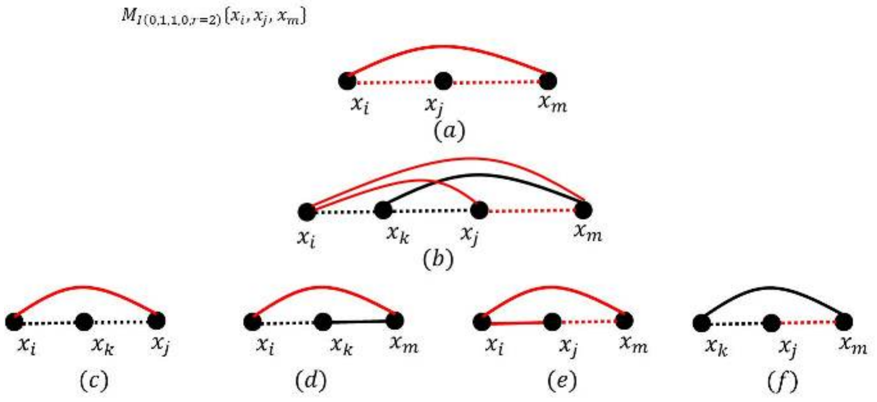


Figure 4.15: $\mathbf{M_{(0,1,1,0,r=2)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{I\{1,0,1,0,r=1\}}}\{x_i, x_j, x_m\}$ (see Figure 4.15).

$$M'_{I\{0,1,1,0,r=2\}}\{x_i,\, x_j,\, x_m\} = max_{i<k<j}\{f(x_i,\, x_k,\, x_j,\, x_m)+$$

$$max\begin{cases} M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\ M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\ M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_k,\, x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} \end{cases}$$

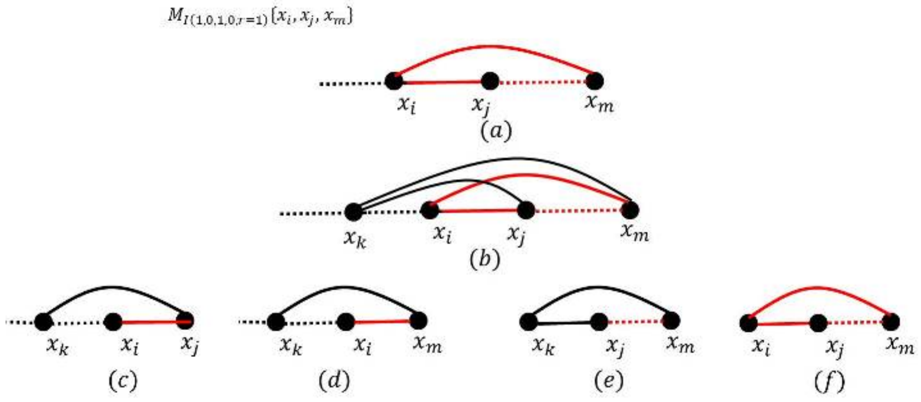


Figure 4.16: $\mathbf{M_{(1,0,1,0,r=1)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{I\{1,0,1,0,r=1\}}}\{x_i, x_j, x_m\}$ (see Figure 4.16).

$$M'_{I\{1,0,1,0,r=1\}}\{x_i,\, x_j,\, x_m\} = max_{k<i}\{f(x_k,\, x_i,\, x_j,\, x_m)+$$

$$max\begin{cases}
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\}
\end{cases}$$

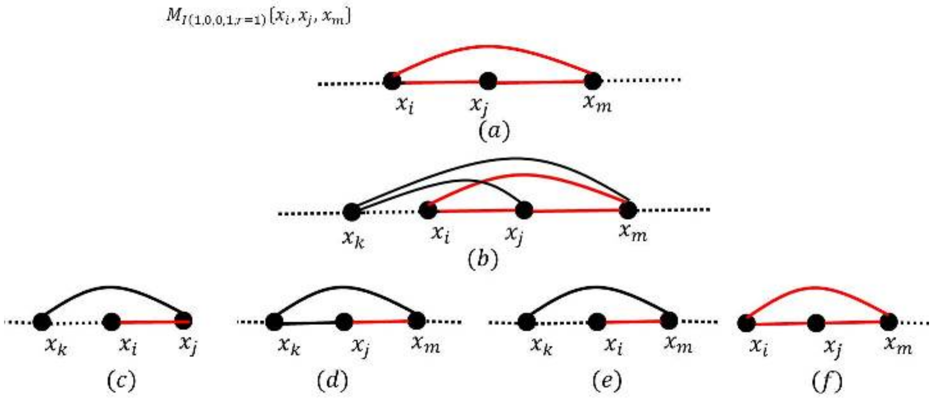


Figure 4.17: $\mathbf{M_{(1,0,0,1,r=1)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{(1,0,0,1,r=1)}}(x_i, x_j, x_m)$ (see Figure 4.17).

$$M_{I(1,0,0,1,r=1)}(x_i,\, x_j,\, x_m) = max_{k<i}\{f(x_k,\, x_i,\, x_j,\, x_m)+$$

$$\begin{cases}
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,1,r=2,4\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,1,r=2,4\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} \\
M'_{I\{1,0,0,1,r=1,4\}}\{x_k,\, x_i,\, x_m\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{1,0,0,1,r=1,4\}}\{x_k,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_j\} & +M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,1,r=1,2,4\}}\{x_k,\, x_i,\, x_m\} & \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} & M'_{I\{1,0,0,1,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_i,\, x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & M'_{I\{0,0,0,1,r=4\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} \\
M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_i,\, x_j\} & M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_j\} & M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} & M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_j,\, x_m\} & M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,0,1,r=4\}}\{x_i,\, x_j,\, x_m\}
\end{cases}$$

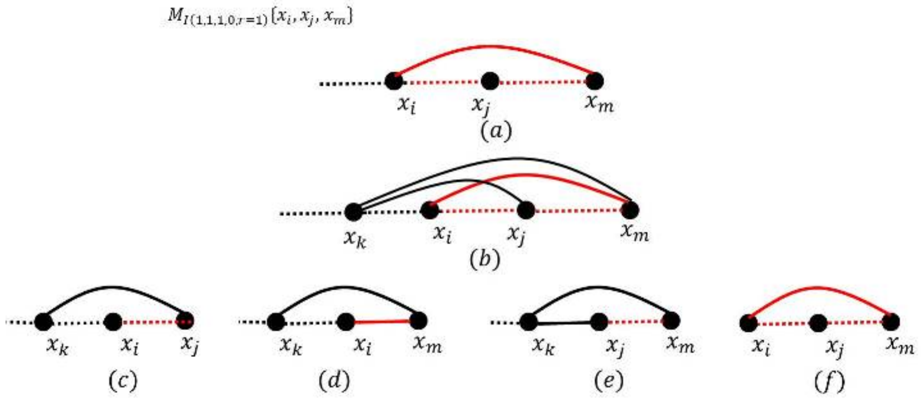


Figure 4.18: $\mathbf{M_{(1,1,1,0,r=1)}}\{x_i, x_j, x_m\}$

Recursive equation for $\mathbf{M'_{I(1,1,1,0,r=1)}}(x_i, x_j, x_m)$ (see Figure 4.18).

$$M'_{I\{1,1,1,0,r=1\}}\{x_i,\,x_j,\,x_m\} = max_{k<i}\{f(x_k,\,x_i,\,x_j,\,x_m)+$$

$$max\begin{cases}
M'_{I\{1,1,1,0,r=1,2,3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_k,\,x_j,\,x_m\} \\
M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\,x_j,\,x_m\} \\
M'_{I\{1,1,1,0,r=1,2,3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{0,0,1,0,r=3\}}\{x_i,\,x_j,\,x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\,x_j,\,x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\,x_i,\,x_m\} & +M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\,x_j,\,x_m\} \\
M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{0,1,0,0,r=2\}}\{x_k,\,x_i,\,x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\,x_j,\,x_m\} \\
M'_{I\{0,0,1,0,r=3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\,x_i,\,x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\,x_j,\,x_m\} \\
M''_{I\{0,1,1,0,r=2,3\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\,x_i,\,x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\,x_j,\,x_m\} \\
M'_{I\{1,0,0,0,r=1\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\,x_i,\,x_m\} + M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\,x_j,\,x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\,x_i,\,x_j\} & +M'_{I\{1,0,0,0,r=1\}}\{x_k,\,x_i,\,x_m\} + M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\,x_j,\,x_m\}
\end{cases}$$

$$\begin{cases}
M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\, x_i,\, x_j\} + M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\, x_i,\, x_j\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_j\} + M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,1,0,r=2,3\}}\{x_k,\, x_i,\, x_j\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1,1,0,0,r=1,2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{0,0,1,0,r=3\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{1,0,1,0,r=1,3\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,0,0,r=2\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{0,1,0,0,r=2\}}\{x_k,\, x_i,\, x_m\} + M'_{I\{1,0,0,0,r=1\}}\{x_k,\, x_j,\, x_m\} + M'_{I\{0,1,1,0,r=2,3\}}\{x_i,\, x_j,\, x_m\} \\
M'_{I\{1000,1\}}\{k,i,j\} + M'_{I\{0100,2\}}\{k,i,m\} + M'_{I\{0010,r3\}}\{k,j,m\} + M'_{I\{0100,2\}}\{i,j,m\} \\
M'_{I\{0100,2\}}\{k,i,j\} + M'_{I\{1000,1\}}\{k,i,m\} + M'_{I\{0010,3\}}\{k,j,m\} + M'_{I\{0100,2\}}\{i,j,m\} \\
M'_{I\{0010,3\}}\{k,i,j\} + M'_{I\{0100,2\}}\{k,i,m\} + M'_{I\{1000,1\}}\{k,j,m\} + M'_{I\{0010,3\}}\{i,j,m\}
\end{cases}$$

These recursive equations slightly modified can also work edge weight, *e.g.*,

Recursive equation for $\mathbf{M'_{I(0,0,1,0,r=3)}}\{x, x_j, x_m\}$

$$M^{'}_{I(0,0,1,0,r=3)}\{x_i,\, x_j,\, x_m\} = max_{j<k<m}\{$$

$$max \begin{cases}
M^{'}_{I(0,0,1,0,r=3)}\{x_j,\, x_k,\, x_m\} & +M^{'}_{I(0,0,1,0,r=3)}\{x_i,\, x_j,\, x_k\} + w\{x_i,\, x_m\} \\
M^{'}_{I(0,1,0,0,r=2)}\{x_j,\, x_k,\, x_m\} & +M^{'}_{I(0,0,1,0,r=3)}\{x_i,\, x_k,\, x_m\} + w\{x_i,\, x_j\} \\
M^{'}_{I(0,1,1,0,r=2,3)}\{x_j,\, x_k,\, x_m\} & +w\{x_i,\, x_j\} + w\{x_k,\, x_i\} + w\{x_i,\, x_m\} \\
M^{'}_{I(0,0,1,0,r=3)}\{x_i,\, x_k,\, x_m\} & +M^{'}_{I(0,0,1,0,r=2)}\{x_i,\, x_j,\, x_k\} + w\{x_k, x_j\}
\end{cases}$$

4.3 Algorithm Analysis

In this section, we will introduce some new notations to introduce a dynamic programming algorithm for the MskT-C and MskT problem on backbone graph, which runs in polynomial time $O(n^{k+1})$ for every fixed k.

Algorithm.

Our algorithm comes from the following observation: every $(k+1)$-clique in any an addition sequence of a k-tree can be obtained by adding a new vertex to an existing k-clique. By the properties in Section 3, the resulted $(k+1)$-clique has at most $k+2$ children and the $(k+1)$-clique consists of $k+1$ number of k-cliques, from which the children, each being a $(k+1)$-clique, can be formed independently and recursively. We present some details in the following.

Let $K = \{x_1, x_2, \cdots, x_k\}$ represent a k-clique and by Proposition 3.5(1) it has at most $k+1$ stretches. Let A_K denote the set of all non-empty stretches for K. For any subset $S \subseteq A_K$, we denote $\mathcal{U}(S) = \bigcup_{s \in S} s$, *i.e.*, the union of all stretches s in S. For any vertex $p \in \mathcal{U}(S)$, we use $K \cup \{p\}$ to denote the $(k+1)$-clique formed by K together with p. This $(k+1)$-clique $K \cup \{p\}$ also results in at most $(k+2)$ stretches by splitting one of the stretches in S into two with vertex p. Let us denote these new set of stretches by $[S, p]$. Also, we define $K_i = K|_p^{x_i}$ for $i = 1, 2, \ldots, k$ and $K_{k+1} = K$ and let $[K, p]$ denote the set $\{K_r \,|\, 1 \le r \le k+1\}$. We then define

$$\Phi(K, S, p) = \{\varphi \,|\, \varphi : [S, p] \to [K, p]\}$$

The stretches in $[S, p]$ can be seen as balls and k-cliques in $[K, p]$ as bins. So, the problem can also be modeled as how many ways exist of throwing $|[S, p]|$ balls into $|[K, p]|$ bins. We observe that every stretch in $[S, p]$ can only be mapped to one of $(k-1)$ k-cliques in $[K, p]$, except the first and the last stretches which can be mapped to one of k k-cliques.

So the functions $\varphi \in \Phi$ are constrained. To be specific, let $K \cup \{p\} = \{x_1, x_2, \cdots, x_{k+1}\}$, with $x_1 < x_2 < \cdots < x_{k+1}$. Then any stretch (ball b_i) with vertices in between x_i and x_{i+1} can only be mapped to some k-clique containing vertices x_i and x_j and the number of such k-cliques is $k-1$.

Let $G = (V, E)$ be an input backbone graph with $V = \{1, 2, \ldots, n\}$. Then for any k-clique $K \subseteq V$, subset of stretches $S \subseteq A_K$, and vertex $p \in \mathcal{U}(S)$, we define $M(K, S)$ to be the maximum objective function value of a k-tree rooted at $(k+1)$-clique $K \cup \{p\}$, for some $p \in \mathcal{U}(S)$. Then we introduce

$$M(K, S) = 0 \qquad \text{if } S = \emptyset \tag{4.1}$$

$$M(K, S) = \max_{p \in \mathcal{U}(\mathcal{S})} \max_{\varphi \in \Phi(K, S, p)} P(K, S, p, \varphi) + w : (K \cup \{p\}) \tag{4.2}$$

$$P(K, S, p, \varphi) = \sum_{\substack{1 \le r \le k+1 \\ K_r \in [K, p],\, \phi^{-1}(K_r) \neq \emptyset}} M(K_r, \varphi^{-1}(K_r)) \tag{4.3}$$

Depending on the objective function above recurrences are formulated for MskT-C, MskT problems such that if it is defined on the edge weight then

$$w : (K \cup \{p\}) = \sum_{e \in (K \cup \{p\}).s.t.\ e\ contains\ p} edge\, weight(e)$$

else it will be defined on the $(k+1)$-clique. For the optimality of the algorithm we want to maximize the following function for for all k-cliques of vertices drawn from V:

$$\max_{K \in [n]^K} M(K, A_K) \tag{4.4}$$

where $[n]^k$ is the set of all k-cliques in V.

Since the algorithm computes the maximum weight of a spanning k-tree rooted at a $(k+1)$-clique formed by some k-clique K. So, recurrence (4.4) will give us the initial step by covering all k-cliques $K \in [n]^k$ with all the possible stretches that K has. The algorithm examines every vertex p from the given allowed stretches S and k-clique K. A new $(k+1)$ clique is formed by adding p to K whose weight contribution is defined as $w : (K \cup \{p\})$. The new set of stretches $[S, p]$ after the addition of p is the same as S, except the stretch in S, from which p was chosen, is split into two by p. This newly $(k+1)$-clique created also has $(k+1)$ number of k-cliques K_r. The function φ maps these new set of stretches to the newly created $k+1$ number of k-cliques K_r, with each such clique using the stretches $\varphi^{-1}(K_r)$ assigned by φ to recursively and independently children $(k+1)$-cliques for the sought spanning k-tree. Recurrence (4.2) maximizes the weight of the spanning k-tree by considering all possible vertices p and functions φ. Recurrence (4.3) branches the computation by independently computing based on every one of the $k+1$ k-clique K_r and sum the total weights in all branches. The algorithm is terminated when there are no stretches available any more, *i.e.*, $S = \emptyset$ shown by Equation (4.1).

In the following paragraphs, we formally prove that the recurrences (4.1)-(4) are correct for MskT-C and MskT.

For convenience we restate Definition 3.2.

Definition 4.3.1. In a spanning k-tree with backbone, children cliques of the same parent are called *duplicate sibings* if they have k-clique in common.

E.g., In the figure 4.19 κ_1, κ_2 and κ_3, κ_4, κ_5 are duplicate siblings. Duplicate siblings will cover different open regions, *e.g*, in κ_1 region in before 4 is covered where as in κ_2 a region b/w 4 and 8 is covered.

Lemma 4.3.2 For any k-tree T there is a k-tree topology for T in which there are no duplicate siblings.

Proof: Let S be an addition sequence for T; then $T(S)$ has this property, by Proposition 3.

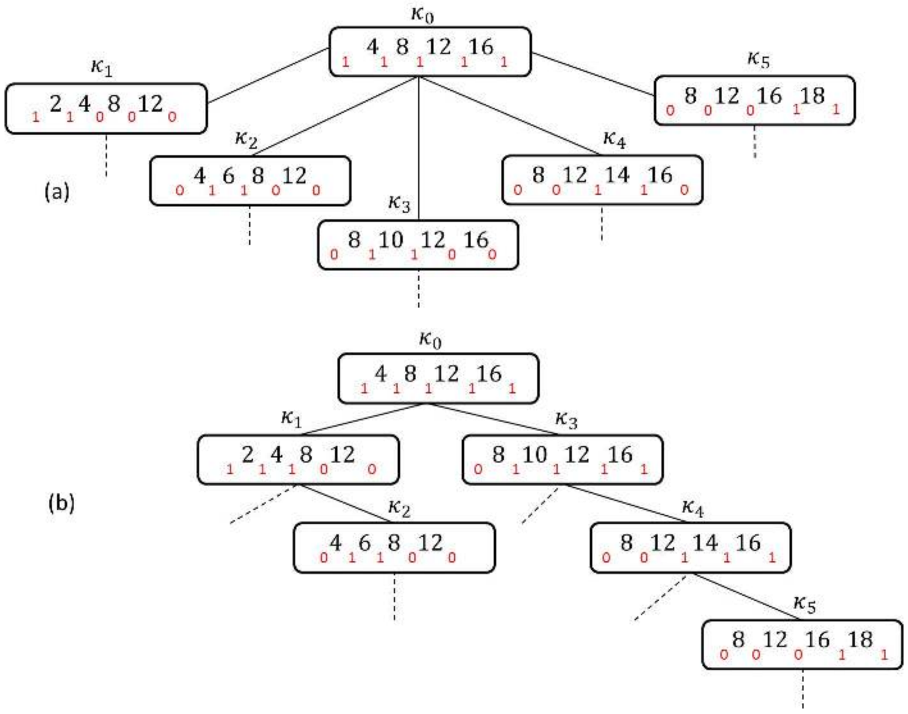


Figure 4.19: illustrates (a) duplicate siblings in a k-tree, and (b) transformation of the k-tree to one without duplicate sibling

Theorem 4.3.3: The algorithm for MSkT examines all spanning k-trees without duplicate siblings.

Proof: Let T be any spanning k-tree without duplicate siblings with addition sequence $\kappa_0, \kappa_1, \cdots, \kappa_m$. Assume base clique $\kappa_0 = \{x_1, x_2, \cdots, x_{k+1}\}$. Then κ_0 can be produced by our algorithm. This is because in equation (4.4) traverses all possible k-clique, $K \in [n]^k$. Thus there must exist one $K = \kappa_0 \setminus \{x_i\}$ for some $1 \leq i \leq k+1$. then in equation (4.2), the algorithm introduces a new vertex x_i as vertex p to form the desired $(k+1)$-clique $K \cup \{x_i\} = \kappa_0$.

Now inductively assume that $(k+1)$-clique κ_i is produced by the algorithm, with importable set I_{κ_i}. So there must exist some K with set of stretches S and there is some vertex $p \in \mathcal{U}(S)$ such that $\kappa_i = K \cup \{p\}$. It is clear that $I_{\kappa_i} = \mathcal{U}([S,\, p])$. Let the children of κ_i be $\kappa_{i_1},\, \kappa_{i_2}, \cdots,\, \kappa_{i_j}$ $(1 \leq j \leq k+1)$. Since T doesn't have duplicate siblings, each k-clique from $\{\kappa_i \cap \kappa_{i_1}, \kappa_i \cap \kappa_{i_2}, \cdots, \kappa_i \cap \kappa_{i_j}\}$ must be unique. By recurrence (4.2) and (4.3), the algorithm maximizes over all possible ways to map the stretches of κ_i to the above k-cliques. In particular, for $l = 1,\, 2, \ldots,\, j$, if $(k+1)$-clique κ_{i_l} has importable set I_{i_l}, through recurrence the algorithm can assign the set of stretches S_{i_l} to k-clique $\kappa_i \cap \kappa_{i_l}$ such that $I_{i_l} = \mathcal{U}([S_{i_l},\, p_{i_l}])$ where $p_{i_l} \in \kappa_{i_l} \setminus \kappa_i$ that can be picked by recurrence. And the algorithm produces $(k+1)$-clique $(\kappa_i \cap \kappa_{i_l}) \cup p_{i_l}$, *i.e.*, κ_{i_l}, as a child of κ_i. Since T doesn't have duplicate siblings, the k-clique from $\{\kappa_i \cap \kappa_{i_1}, \kappa_i \cap \kappa_{i_2}, \cdots, \kappa_i \cap \kappa_{i_j}\}$ are different. Therefore, the mapping functions in φ guarantee that $I_{\kappa_l} \cap I_{\kappa_t} = \emptyset$ for $1 \leq l,\, t \leq j$ and $l \neq t$, a necessary condition for importable sets stated in Proposition 3.9.

Claim: Running time of the algorithm is $O(n^{k+1}(k+1)^{k+2}1.44^k)$.

As Any k-clique can be the root clique so we traverse all the k-clique in $[n]^k$ needs $O(n^k)$ time in equation (4.4). So for as fixed k-clique K with S By Lemma 3.5 it has at most $(k+1)$ available stretches. In equation (4.2), the maximization over p in these $\mathcal{U}(S)$ takes $O(n)$ time. These give the $O(n^{k+1})$ polynomial factor. Since $\kappa = K \cup \{p\}$ can be expressed as combination of $(k+1)$ k-cliques K_r where each $\varphi^{-1}(K_r)$ cover stretches. When the stretches in $|[S,\, p]| =(k+2)$ for $K \cup \{p\}$ this would be like throwing $(k+2)$ different balls into $[K,\, p]$ different bins. As functions $\varphi \in \Phi$ are constrained because of the observation that every stretch in $[S,\, p]$ can only be mapped to one of $(k-1)$ k-cliques in $[K,\, p]$, except the first and the last stretches which can be mapped to one of k k-cliques. So the $|[S,\, p]| =(k+2)$, $|\Phi| = k^2(k-1)^k$;

For the same k-clique K, $K \cup \{p\}$, $|[S,\, p]| =(k+1)$, then $(k+1)$ open regions can be chosen from $(k+2)$ possible regions, the number of such cases will be $\binom{k+2}{k+1}$ and we have the similar question to ask: how the $(k+1)$ are distributed into are distributed into $(k+1)$ k-cliques. By the above analysis, the total number such cases, *i.e.*, $|\Phi| \leq \binom{k+2}{k+1} k^2(k-1)^{k-1}$.

Similarly, for the same K if $|[S,\, p]| = k$; The number of cases overall is $\leq \binom{k+2}{k} k^2 (k-1)^{k-2}$.

Following the same analysis, the total number of cases for any k-clique K is

$$k^2(k-1)^k + \binom{k+2}{k+1} k^2(k-1)^{k-1} + \binom{k+2}{k} k^2(k-1)^{k-2} + \cdots + \binom{k+2}{1} k$$

$$\begin{aligned} &< \; k^{k+2} + \binom{k+2}{k+1} k^{k+1} + \binom{k+2}{k} k^k + \cdots + \binom{k+2}{1} k \\ &= \; (k+1)^{k+2}. \end{aligned}$$

The summation in recurrence (4.3) takes $O(k)$ time. Therefore, the running time of the algorithm is $O(k(k+1)^{k+2} n^{k+1})$. Also derived from the above analysis is the space complexity $O(2^{k+1} n^k)$.

Chapter 5

Hardness

We would like to know if there are polynomial algorithms for the problem MskT and MskT-C , with a polynomial degree independent of k or a polynomial degree d for some $d < k+1$ (d may be a function of k). We prove problem MskT-C remains NP-hard. In particular, We construct a reduction from the k-Clique problem to MskT-C that essentially preserves the parameter k.

For any $(k+1)$-clique $\{x_1, x_2, \cdots, x_k, x_{k+1}\}$ in $G = (V, E)$ with weight $w : E \rightarrow R_{\geq 0}$, we define function

$$\lambda(\mathcal{E}(x_1, x_2, \ldots, x_k, x_{k+1}))$$

where $\mathcal{E}(x_1, x_2, \ldots, x_k, x_{k+1}) = \langle w(x_1, x_2), w(x_1, x_3), \ldots, w(x_k, x_{k+1})\rangle$, i.e., the m-tuple of weights of all edges in the $(k+1)$-clique $\{x_1, x_2, \ldots, x_k, x_{k+1}\}$, for $m = \frac{1}{2}k(k+1)$, and λ is a mapping: $\lambda : \bigcup_{m=1}^{\infty} R^m_{\geq 0} \rightarrow R_{\geq 0}$.

Problem Decision MskT-C:

Input: integer $k \geq 1$, backbone graph $G = (V, E)$, weight $w : E \rightarrow R_{\geq 0}$ and $W > 0$.

Question: Does G have a spanning k-tree $H = (V, F)$ such that $\sum_{\tau \in H} \lambda(\mathcal{E}(\tau)) \geq W$?

where $\tau \in H$ is to represent the statement that $(k+1)$-clique τ belongs to k-tree H.

Theorem 5.1: The problem Decision MskT-C is NP-complete.

Proof: Given a spanning k-tree H, it is not difficult to see that it will take polynomial time to determine if the defined objective function value is at least W. Therefore, the problem is in NP.

The hardness is proved with a reduction from k-CLIQUE. Given a graph $G = (V, E)$, where $V = \{x_1, x_2, \ldots, x_n\}$ as an instance of k-CLIQUE, we construct an instance of complete backbone graph $G' = (V', E')$ as follows in polynomial time:

1. $V' = \{1, 2, \ldots, n\}$;
2. $E' = \{\{i, j\} : i < j \text{ and } i, j = 1, 2, \ldots, n\}$;
3. backbone $B = \{\{i, i+1\} : i = 1, 2, \ldots, n-1\}$;
4. $k' = k - 1$; $W = 1$;
5. weight function $w(i, j) = 1$ if $(x_i, x_j) \in E$; $w(i, j) = 0$ otherwise.

Now consider a simple function $\lambda : \bigcup_{m=1}^{\infty} R_{\geq 0}^m \to R_{\geq 0}$ such that for any $m \geq 1$ and $\langle e_1, e_2, \cdots, e_m \rangle \in R_{\geq 0}^m$,

$$\lambda(\langle e_1, e_2, \ldots, e_m \rangle) = 1 \text{ if and only if } e_i > 0, \forall i \, 1 \leq i \leq m.$$

For given $k \geq 3$, if the constructed G' has a spanning k'-tree H with objective function value at least 1, where $k' = k - 1$, the contribution of the value must have come from at least one (k')-clique in the spanning tree. Let this clique be $\kappa = \{i_1, i_2, \ldots, i_k\}$ in H. Since the function w' on the clique has value 1 if and only if every $e \in E'(\kappa) \subseteq E$, Based on the construction of G', these edges are among the vertex set $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ in the original graph G. Thus G has a clique of size k.

On the other hand, if G has a clique of size k, then G' has a spanning k'-tree H of objective function value $\geq W = 1$. To see this, let the clique of size k in G be $\{x_{i_1}, x_{i_2}, \cdots, x_{i_k}\}$. Let k-clique $\kappa_0 = \{i_1, i_2, \ldots, i_k\}$. We show in the following that there is a spanning k-tree rooted at κ_0 for G'.

Without loss of generality, we assume $i_1 < i_2 < \cdots < i_k$ in κ_0. By section 3, κ_0 can have s non-empty stretches, for $s \leq k+1$. Then on backbone graph G' a $(k-1)$-tree H rooted at κ_0 with at most s branches can be constructed with each branch covering a stretch. In particular, for every $j = 0, 1, \ldots, k$, the jth stretch is $\{i_j + 1, i_j + 2, \ldots, i_{j+1} - 1\}$. Let $m_j = i_{j+1} - i_j - 1$, the length of the jth stretch. If $m_j \neq 0$, the jth stretch yields a branch of k-cliques represented by the following k-clique sequence: $\kappa_0, \kappa_{j_1}, \kappa_{j_2}, \ldots, \kappa_{j_{m_j}}$, where $\kappa_{j_1} = \kappa_0|_{i_j+1}^{i_j}$, $\kappa_{j_2} = \kappa_{j_1}|_{i_j+2}^{i_j+1}$, $\ldots, \kappa_{j_{m_j}} = \kappa_{j_{m_j-1}}|_{i_j+m_j}^{i_j+m_j-1}$ (note that $i_j + m_j = i_{j+1} - 1$). Details of boundary cases $j = 0$ and $j = k$ are only slightly different from the general case and thus are omitted. H covers all backbone edges of G' and is a spanning k-tree for G'. In addition, since $w'(\kappa_0) = 1$, the weight of this $(k-1)$-tree is $\geq w(\kappa_0) \geq W$.

The proof of the above Theorem constructs a reduction that is actually also a parameter-preserving polynomial time reduction between the two parametrized problems.

Because k-clique is W[1]-complete [56] and it cannot be solved in time $f(k)n^{o(k)}$ for any function f unless W[1]=FPT [57], we conclude

Corollary 5.1. Decision MskT-C is $W[1]$-hard.

Theorem 5.2. Unless the W-hierarchy collapses, the MskT-C problem cannot be solved in time $f(k)n^{o(k)}$ for any function f.

Chapter 6

K-trees as Stochastic Context Sensitive Grammars

In formal language systems any generation process of a string is some sequence of syntactic rules applied and thus yields a syntactic structure associated with (the objects on) the string. In stochastic formal language there is more than one syntactic process to generate the same string [57, 58]. A probability distribution is associated with rules. If we sum up the probabilities of those rules which are used in that generation process of a string then we will get the probability for the specific syntactic structure admitted by the string. Therefore, a stochastic formal language system defines a probability space for all the strings in a defined language. At the same time, it defines, for every given string, a probability space for all the syntactic structures admitted by the string.

A biomolecule consists of a string of linearly arranged residues that can spatially interact to fold the sequence into a three-dimension structure. Such a structure can be predicted as a syntactic structure of the molecule string through statistical computation with the stochastic context free grammar model. The recursive context-free rules enable dynamic programming algorithms (of running time $O(n^3)$) efficient for structure decoding (CYK), probability computation (Inside) with SCFG models and SCFG model learning algorithms, *e.g.*, Inside-Outside [46, 59].

The capability of SCFG, however, is limited to modeling context-free structures, which include "nested" and "parallel" relationships of objects on a linear string (Figure 6.1 (a)). With SCFG, it is not possible to model objects arbitrarily "crossing each other" (Figure 6.1(b)), which also often arise from important applications. This is typically the case in RNA pseudoknot prediction. Theoretically, such crossing relationships are characterizable with formal context-sensitive grammars, but general context-sensitive rules are less convenient to use and incur computation intractability [57, 59].

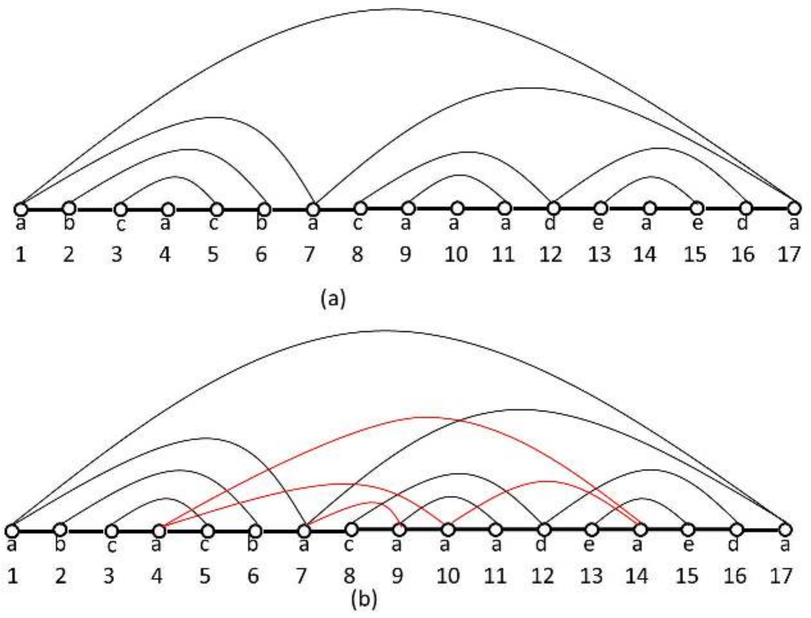


Figure 6.1: (a) “Parallel” and “nested” relationships between letters on string `abcacbacafadeaeda` of length 17, characterizable with SCFG; (b) “crossing” relationships as the result of including additional (red color) relationships, which cannot be modeled with SCFG.

Backbone k-trees make it possible to introduce a new schochastic linguistic modelas as a natural extension from SCFG that can characterize mildly context-sensitive structures. k-trees have the ability to model constraint “crossing” relationships among the objects on a string. Such constrained context-sensitive relationships can be found in a number of important applications. Typically our studies have revealed that residue relationship graphs formed by resolved biomolecule 3D structures have treedecompositions with treewidth typically $k = 3$ or 4 (occasionally higher). Such applications have motivated our current work to seek feasible probabilistic modeling of definable over linear chains of objects (*i.e.*, strings).

6.1 Relationship of SCSGs with Backbone k-trees

Previous studies showed that learning and statistical analysis problems over general networks are extremely difficult, for instance NP-hard even for $k = 2$, thus excluding the possibility to establish feasible implementation of such a framework [60]. However, with the linear chain constraint on k-trees, we are able to show, for the first time, that these problems are feasible, namely solvable in polynomial-time for every fixed value of k. Here we will be considering a generalized rooted tree topology for a rooted k-tree where each non-base clique κ_j is assigned a unique parent κ_i such that $\kappa_j = \kappa_i|_y^x$ for some $x \neq y$. Note that duplicate siblings are allowed in a generalized rooted tree topology.

6.1.1 Faithful k-trees and backbone k-trees

Now we introduce k-trees for strings of a language. Let Σ be finite alphabet for language and $s \in \Sigma^*$ a string. By $|s|$, we denote the length of s, *i.e.*, the number of symbols in string s. Assume $|s| = n$; we use $[n] = \{1, 2, \ldots, n\}$ for the set of indexes of all symbols in s and $s(i)$ for the ith symbol, $i \in [n]$.

Definition 6.1.1.1 Let s be a string of length n. A (labeled) graph $G = (V, E)$ is *faithful* to s if

1. $V \subseteq [n]$;

2. $\forall\, i \in V$, vertex i is labeled with $s(i)$; and

3. $\forall\, i, j \in V$, if $i < j$ and $\neg\exists l \in V$ such that $i < l < j$, then $(i, j) \in E$.

A labled k-tree satisfying the above three conditions is called a k-tree *faithful to s*.

Proposition 6.1.1.2 Given any string s of length n, the set of all k-trees faithful to s are inductively defined as follows:

1. A $(k + 1)$-clique of vertices in $[n]$ faithful to s is a k-tree faithful to s;

2. Let $G = (V, E)$ be a k-tree faithful to s and $y \in [n] - V$. Then for any k-clique C in G containing the highest index (in G) below y (if it exists) and the lowest index (in G)

above y (if it exists), the graph

$$G' = (V',\, E'),\ \text{where } V' = V \cup \{y\}\text{and } E' = E \cup \{(y,\, i) \,|\, i \in C\})$$

is a k-tree faithful to s.

We are interested in k-trees faithful to s which cover all vertices in $[n]$ to characterize structures imposed on the whole string s. For convenience, we call a k-tree faithful to the string s a *backbone k-tree* for s if it includes exactly all vertices in $[n]$. Figure 6.2(a) shows a 3-tree faithful to a string s of length 7, (b) a backbone 3-tree for s, and (c) a 3-tree but not faithful to s.

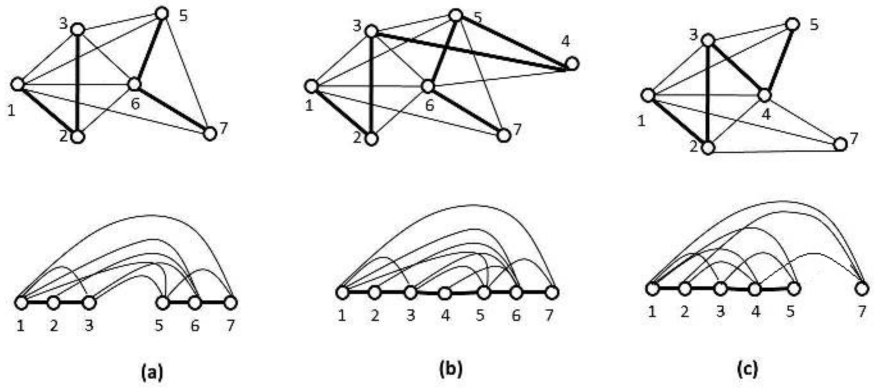


Figure 6.2: (a) A 3-tree faithful to a string of length 7, (b) after vertex 4 is added to the k-tree in (a), it becomes a backbone 3-tree for the string, and (c) a 3-tree but not faithful to the string. Bold indicates backbone edge.

Proposition 6.1.1.3 Let $G = (V,\, E)$ be a k-tree faithful to s of length n, where $V = \{v_1,\, v_2, \ldots,\, v_m\}$ for some $m < n$ such that $v_i < v_{i+1}$ for all $i = 1,\, 2, \ldots,\, m-1$. Let $v \in [n] - V$. Assume $v_j < v < v_{j+1}$ for some $v_j,\, v_{j+1} \in V$. Then creating a new k-tree faithful to s by connecting v to G satisfies the following constraints:

1. $\exists\, U \subseteq V$, a k-clique, such that v forms a $(k+1)$-clique with U, and

2. $v_j,\, v_{j+1} \in U$.

6.2 CSGs for 2-trees

In this section we give an idea towards a formal formulation of CSG for 2-trees, which may be extended to the CSGs of general k-trees. For any alphabet Σ let $G(\Sigma)$ be the grammar with terminal set Σ, non-terminal set $\{S_0,\ I,\ P,\ S,\ I',I'',\ L,\ R\}$, start symbol S_0, and the set of rules which are obtained from schemata 1-9 below by all possible substituions of symbols in Σ for the variable x, y and z. We also allow any of the non-terminals in the body of the rule to be omitted, as if each of them can be nullified. So for example schema 2 generates a total of $4 \cdot |\Sigma|^3$ rules in $G(\Sigma)$.

1. $S_0 \rightarrow PxIyS|\,LxI'yS|\,PxI''yR|\,LxI'zI''yR$
2. $xIy \rightarrow xIzIy$
3. $PxIy \rightarrow PzIxIy|\,LzI'xIy|\,P'zy$
4. $P'zy \rightarrow PkIzy|\,LkIzy$
5. $xIyS \rightarrow xIyIzS|\,xIyI''zR$
6. $LxI'y \rightarrow LxI'zIy|\,PxIzIy$
7. $xI'yS \rightarrow xI'yIzS|\,xI'yI''zR$
8. $PxI''y \rightarrow PzIxI''y|\,LxIzI''y$
9. $xI''yR \rightarrow xIzI''yR|\,xIzIyS$

We present an example of generating a string $ACGUAC$ with alphabets $\Sigma = \{A,\ C,\ G,\ U\}$. Figure 6.3 shows the syntatic structure admittied by this generation process.

1. $S_0 \rightarrow PCI''AR$ *rule 1.1*

2. $\rightarrow PAICI''AR$ *rule 7.1*

3. $\rightarrow AICI''AR$ P *empty*

4. $\rightarrow ACI''AR$ I empty

5. $\rightarrow ACIUI''AR$ *rule 8.1*

6. $\rightarrow ACIGIUI''AR$ *rule 2*

7. $\rightarrow ACGIUI''AR$ I *empty*

8. $\rightarrow ACGUI''AR$ I *empty*

9. $\rightarrow ACGUACS$ *variant of rule 9.2*

10. $\rightarrow ACGUAC$ S *empty*

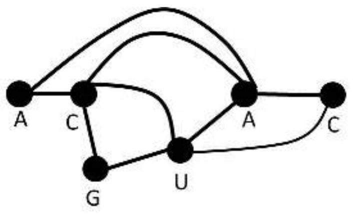


Figure 6.3: One possible syntactic structure of 2-tree for string $ACGUAC$

6.3 SCS Language Models

6.3.0.1 Probability distributions of strings

We represent any k-tree G with a pair $G = (\mathcal{B}_G, \mathcal{T}_G)$, in which $\mathcal{B}_\mathcal{G}$ is the collection of all $(k+1)$-cliques in G and $\mathcal{T}_G$ is a tree topology connecting these $(k+1)$-cliques. Figure 6.4(a) illustrates generation of a 3-tree; and (b) gives a tree topology representation in (a).

Given a string, there may be more than one associated backbone k-tree. Proposition

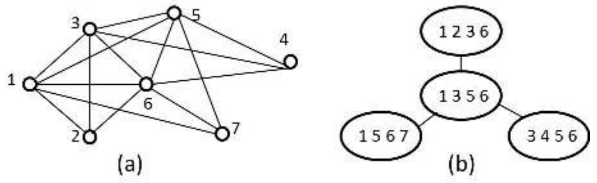


Figure 6.4: (a) Illustration for generation of a 3-tree with 7 vertices (b) Tree topology representation of the 3-tree in (a).

6.1.1.2 provides a nondeterministic processes to generate faithful k-trees and backbone k-trees. Different backbone k-trees for the same string correspond to different processes to generate the string, thus cfunctionorresponding to different possible syntactic structures for the string. We model such structures statistically.

Let $\kappa = \{v_1, v_2, \ldots, v_{k+1}\}$ be a $(k+1)$-clique in any k-tree faithful to some string s. For convenience, we assume the vertices are always ordered increasingly as $1 \leq v_1 < v_2 < \ldots v_{k+1} \leq n$. We associate κ with two pieces of information:

1. $C_\kappa = \langle s(v_1), s(v_2), \ldots, s(v_{k+1})\rangle$;

2. $D_\kappa = \langle v_2 - v_1, v_3 - v_2, \ldots, v_{k+1} - v_k\rangle$.

Two $(k+1)$-cliques κ_1 and κ_2 are called *equivalent* if both $C_{\kappa_1} = C_{\kappa_2}$ and $D_{\kappa_1} = D_{\kappa_2}$ hold. We denote the class of equivalent $(k+1)$-cliques to given κ by $\mathcal{E}_\kappa$. Thus the space of $(k+1)$-cliques is partitioned into exclusive classes of equivalent $(k+1)$-cliques with a probability distribution. Such a distribution can be estimated as follows from a sufficiently large population of backbone k-trees for strings which is presumably available.

Note: Let $t \geq 1$ be a threshold, which we treat as a parameter to be determined by the availability of the data. κ_1 is equivalent to κ_2 iff $C_{\kappa_1} = C_{\kappa_2}$ and for each i, $1 \leq i \leq k$, the ith member of $D_{\kappa_1}$and D_{κ_2} are equal or both $> t$.

Let G be a k-tree faithful to some string and $\mathcal{T}_G$ be a representation of G as a rooted k-tree topology. We will define the probability of $\mathcal{T}_G$ inductively.

First, in the atomic case that G is a $(k+1)$-clique, the probability $p(G)$ of G is the ratio of the number of occurrences of $(k+1)$-cliques in $\mathcal{E}_G$ over the number of occurrences of all $(k+1)$-cliques.

Second, in the case of a more general k-tree topology $\mathcal{T}_G$, we can compute the probability of a $\mathcal{T}_G$ as follows based on the inductive generation of k-trees given in Proposition 6.1.1.2. Assume κ to be the base clique of $\mathcal{T}_G$, with m children $\kappa_1, \ldots, \kappa_m$, for some $m \geq 1$, in tree $\mathcal{T}_G$. For $i = 1, 2, \ldots, m$, let $\mathcal{T}_{G_i}$ be the k-tree topology for child clique κ_i and its descendents in $\mathcal{T}_G$.

Then the probability $p(\mathcal{T}_G)$ is computed as

$$p(\mathcal{T}_G) = p(\kappa, \mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m}) = p(\kappa | \mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m}) p(\mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m})$$

in which, due to the independence of k-tree topologies $\mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m}$,

$$p(\mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m}) = p(\mathcal{T}_{G_1}) p(\mathcal{T}_{G_2}) \ldots p(\mathcal{T}_{G_m})$$

and the independence of separated components in a k-tree,

$$p(\kappa | \mathcal{T}_{G_1}, \mathcal{T}_{G_2}, \ldots, \mathcal{T}_{G_m}) = p(\kappa | \kappa_1, \kappa_2, \ldots, \kappa_m)$$

where the relationships among κ and κ_i, for all $i = 1, 2, \ldots, m$, satisfy the Parent-Children relationship. We call this set of $(k+1)$-cliques $\{\kappa, \kappa_1, \kappa_2, \ldots, \kappa_m\}$ a k-tree *m-claw*. Then the probability $p(\kappa | \kappa_1, \kappa_2, \ldots, \kappa_m)$ is evaluated from a sufficiently large database to be the frequency of k-tree m-claws equivalent with the k-tree m-claw

$$\{\kappa, \kappa_1, \kappa_2, \ldots, \kappa_m\}.$$

That is the ratio of the number of occurrences of such k-tree m-claws over the total number of occurrences of k-tree m-claws .

Now we are ready to define probability distributions for strings of languages. For the convenience of discussion, we use the notation $BbkT(\mathcal{T}_G,\, s)$ for the predicate asserting that $\mathcal{T}_G$ is a tree topology of a backbone k-tree G for string s. Let θ be the probability distribution drawn from some data source for atomic k-trees and k-tree m-claws. Then the aforementioned probability $p(\mathcal{T}_G)$ for k-tree topology $\mathcal{T}_G$ should be conditional $p(\mathcal{T}_G|\theta)$.

Distribution θ defines a language L_θ as follows.

$$s \in L_\theta \Longleftrightarrow \exists \mathcal{T}_G \; BbkT(\mathcal{T}_G,\, s) \,\&\, p(\mathcal{T}_G|\theta) > 0$$

Therefore, the membership of s can be determined by checking whether

$$functionProb(s|\theta) = \sum_{BbkT(\mathcal{T}_G,\, s)} Prob(s, \mathcal{T}_G|\theta) = \sum_{BbkT(\mathcal{T}_G,\, s)} p(\mathcal{T}_G|\theta) > 0.$$

$Prob(s|\theta)$ is called the *total probability* of s being produced by the defined system θ, or of belonging to the language L_θ.

Often, instead of computing the probability of all possible tree topologies of backbone k-tree for s, we may need to find a tree topology for s which is the most likely. Such a topology corresponds to the most “plausible” syntactic structure of the string s. In particular, the following formulates the *most probable topology of backbone* k-tree for s:

$$G^* = \arg \max_{BbkT(G,s)} p(G|\theta)$$

In the next section, we will present efficient algorithms developed to compute both total and maximum probabilities of any given string s.

6.3.0.2 Algorithms

Technically, a k-tree topology faithful to string s covers some (possibly discontiguous) vertices ordered from 1, 2 to $n = |s|$. Let G be such a k-tree topology. Then it can be specified with two pieces of information $\langle R_G, I_G \rangle$, where R_G is a $(k+1)$-clique, the base of the tree topology corresponding to G, and I_G is an *importable vector*:

$$I_G = (b_0, b_1, \dots, b_{k+1}), \;\; b_i \in \{0, 1\}, \, i = 0, 1, \dots, k+1$$

where, assume $R_G = \{v_1, v_2, \dots, v_{k+1}\}$ with $1 \leq v_1 < v_2 < \cdots < v_{k+1} \leq n$, the boolean indicator b_i indicates if all vertices between v_i and v_{i+1} are included in G, for $0 \leq i \leq k+1$ ($v_0 = 1$ and $v_{k+2} = n$ by default).

We obtain important relationships of importable sets associated with k-tree topologies. Let $(k+1)$-clique R_{G_1} be a child of R_G in the tree topology $\mathcal{T}_G$ and R_{G_1} is the root of the tree topology of some k-tree G_1 contained in G. Let $I_G = (b_0, b_1, \dots, b_{k+1})$. Then there exists some i and j, either $j < i - 1$ or $j \geq i + 1$, such that

1. $b_j = 1$, and

2. $R_{G1} = R_G|_y^{v_i}$ for some y, $v_j < y < v_{j+1}$.

In addition, if $j \geq i + 1$, then

$$I_{G_1} = (b_0, \dots, b_{i-1}, 0, b_{i+2}, \dots, b_{j-1}, b'_j, b''_j, \dots, b_k)$$

if where $b'_j = 1$ unless $v_{j-1} = y - 1$, and $b''_j = 1$ unless $v_j = y + 1$. The case $j \leq i - 1$ is similar. If in addition, R_{G_2}, the root of tree topology for k-tree G_2, is another child of R_G in the tree topology, with $R_{G_2} = R_G|_z^w$, for some $w \in R_G$, then

3. I_{G_2} has 0 in its jth component.

The above conditions (1)-(3) constrain R_G to have at most $k+2$ children.

We now establish recurrences for probabilities of k-tree topologies faithful to s. Let G be such a k-tree topology with base R_G and importable vector I_G. Let the total probability for s to admit G (with the importable vector I_G) be $t(R_G,\, I_G)$. Then

$$t(R_G,\, I_G) = \sum_{m=1}^{k+2} \prod_{pc(R_G,\, I_G,\, R_{G_i},\, I_{G_i}),\, i=1}^{m} t(R_{G_i},\, I_{G_i}) \times p(R_G | R_{G_1}, \ldots,\, R_{G_m}) \tag{6.1}$$

where predicate $pc(R_G,\, I_G,\, R_{G_i},\, I_{G_i})$ asserts that $(k+1)$-clique R_G is the parent of R_{G_i} in the tree topology for G, given their importable sets. The above probability includes all possible situations of R_G having m children, $m \leq k+2$. The probability $p(R_G | R_{G_1}, \ldots,\, R_{G_m})$ was defined in the previous section.

The base case of $t(R_G,\, I_G)$ is when R_G has no child with $I_G = \emptyset$, which is the case that G is simply a $(k+1)$-clique. Therefore $t(R_G,\, \emptyset) = p(R_G)$, which is also defined in the previous section.

The recurrence (6.1), together with the base case, offers a dynamic programming algorithm for computing the total probability $t(R_G,\, I_G)$. This is to establish a table with entries for storing $t(R_G,\, I_G)$ for all k-tree topologies G. The table has dimensions $O(n^{k+1}) \times 2^{k+2}$ since there are $O(n^{k+1})$ different $(k+1)$-cliques in total. Each entry needs time $O(k^2 n)$ to compute, with total time complexity $O(n^{k+2} 2^{k+2} k^2)$. For a fixed value of k the time complexity is $O(n^{k+2})$. For small k the hidden constant should be small enough to allow for practical implementations of the algorithm.

The total probability of s is then defined as

$$Prob(s|\theta) = \sum_G t(R_G,\, I^*)$$

where importable vector $I^* = (1,\, 1, \ldots,\, 1)$ is of all 1's, meaning to include all vertices

$\{1, 2, \ldots, n\}$.

Often, instead of computing the total probability of s, it is desirable to know the most likely syntactic structure the string s may possess. This is to compute the maximum probability of a backbone k-tree topology faithful to s:

$$G^* = \arg \max_{BbkT(G,\, s)} p(G|\theta)$$

For this purpose, we define $z(B,\, I)$ to be the maximum probability of a k-tree topology faithful to s with base B (and importable vector I). Then we have a recurrence resembling (6.1) in which the summation is replaced by maximization:

$$z(B,\, I) = \max_{m=1}^{k+2} \prod_{pc(B,\, I,\, B_i\, ,I_i),\, i=1}^{m} z(B_i,\, I_i) \times p(B|B_1, \ldots,\, B_m). \tag{6.2}$$

The base case for $z(B,\, I)$ is when $I = \emptyset$ and the corresponding k-tree is simply B, a $(k+1)$-clique, resulting in $z(B,\, \emptyset) = p(B)$.

The backbone k-tree G^* for s corresponding to B such that $z(B,\, I^*)$ achieves the maximum value in the table. In this case, $B = R_{G^*}$.

Finally, it is not difficult to see that, like computing $p(B,\, I)$, computing $z(B,\, I)$ can be done in dynamic programming as well with same asymptotic computational time requirement.

If the effect of tree topologies with duplicate siblings can be neglected then the non trivial algorithm presented in the Chapter 4 can be adapted to calculate $p(B,\, I)$ and $z(B,\, I)$. The table for this algorithm has dimensions $O(n^k) \times 2^{k+1}$ since there are $O(n^k)$ different k-cliques in total. Each entry needs time $O((k+1)^{k+2}n)$ to compute, with total time complexity $O(k(k+1)^{k+2}n^{k+1})$.

Conclusion

We introduced a polynomial time algorithm for finding the maximum spanning k-trees on the family of backbone graphs. There is a strong evidence that the algorithm has the optimal order of growth in input size n for each k greater than 2. The reason is that, backbone spanning k-trees characterize graph relationships among linearly connected vertices (on the backbone), and they are also applicable to relationships between symbols on sentences of formal languages. Then the algorithm introduced in section 4 implies that mildly context-sensitive languages can be syntactically defined and their sentences can be parsed in time $O(n^{k+1})$ for every fixed k. The family of languages for $k = 2$ includes all context-free languages as a special case. Parsing of general context-free languages can be accomplished in $O(n^3)$ [61, 62]. A faster algorithm (*i.e.*, of time $O(n^{k+1-\epsilon})$, for any $\epsilon > 0$) for Maximum Spanning k-Tree would imply a sub-cubic algorithm to parse general context-free languages, breaking the longstanding barrier of $\theta(n^3)$ for practical algorithms.

The recursive rules for backbone k-tree generation have allowed us to associate probability distributions with k-trees in a natural way. This leads to efficient dynamic programming algorithms for probability computation of backbone k-trees with a stochastic context sensitive language model, suitable for statistical analysis of real-world structures that can be modeled upon strings of a language. The probability computation algorithms we developed are in the spirit of CYK and Inside algorithms (as with SCFG), we are optimistic that there will emerge EM learning algorithms for the newly introduced model in the spirit of Inside-Outside (as with SCFG).

References

[1] C. Laing, T. Schlick. Computational approaches to RNA structure prediction, analysis and design. *Curr Opin Struct Biol*, 21(3)306-318, June 2011.

[2] W.R. Taylor, C. A. Orengo. Protein structure alignment. *Journal of Molecular Biology*, 208(1)1-22, 5 July 1989.

[3] R.B. Lyngsø. Computational Biology: Ph.d. Dissertation BRICS, Computer Science Department, University of Aarhus, 2000, 173 pages.

[4] J. Xu, F Jiao, B. Berger. A Tree-Decomposition Approach to Protein Structure Prediction. *Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference*, Pages 247-256.

[5] R.H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering*, 7:1059-1068, 1994.

[6] T. Akutsu, S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210:261-275, 1999.

[7] N. A. Pierce, E. Winfree. Protein design is NP hard. *Protein Engineering*, 15(10):779-782, 2002.

[8] T. Akutsu. NP-hardness results for protein side-chain packing. *Genome Informatics*, 8:180-186, 1997.

[9] A. Sali, E. Shahknovich, and M. Karplus. How does a protein fold ? *Nature*, 369:248-251, 1994.

[10] Tinoco I Jr, Bustamante C. How RNA Folds. *J Mol Biol*, 22;293(2):271-81, Oct 1999.

[11] R.B. Lyngsø. Complexity of pseudoknot prediction in simple models (2004), *Automata, Languages and Programming Lecture Notes in Computer Science*,3142:919-931, 2004.

[12] Bern, Marshall Wayne. NETWORK DESIGN PROBLEMS: Steiner Trees and Spanning K - Trees. University of California, Berkeley, 1987, 148 pages.

[13] Leizhen Cai and FrCdkric Maffray. On the SPANNING k-TREE problem. *Discrete Applied Mathematics,* 44:139-156, 1997 North-Holland.

[14] Yinglei Song, Chunmei Liu, Xiuzhen Huang, Russell L. Malmberg, Ying Xu, and Liming Cai. Effecient parameterized algorithms for biopolymer structure-sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:423-432, 2006.

[15] Lorena Nasalean, Jesse Stombaugh, Craig L. Zirbel, and Neocles B. Leontis. RNA 3D Structural Motifs: Definition, Identification, Annotation, and Database Searching Non-Coding RNAs. *Springer Series in Biophysics,* 13:1-26, 2009.

[16] Ruth Nussinov, George Piecznik, Jerrold R Griggs, and Daniel J Kleitman. Algorithms for loop matching. *SIAM Journal on Applied Mathematics*, 35(1):68-82, 1978.

[17] M. Zuker. Computer prediction of RNA structure. *Methods in Enzymology*, 180:262-288, 1989.

[18] I.L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31:3429-3431, 2003.

[19] M. Zuker. Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, 31(13):3406-3415, 2003.

[20] M. Zuker and P. Steigler. Optimal computer folding of larger RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9:133-148, 1981.

[21] B Knudsen and J Hein. RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15(6):446-454, 1999.

[22] Tadao Kasami. An effcient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab, Bedford, MA, 1965.

[23] Elena Rivas and Sean R. Eddy. Secondary structure alone is generally not statistically signicant for the detection of noncoding rnas. *Bioinformatics*, 16(7):583-605, 2000.

[24] Chuong B. Do, Daniel A. Woods, and Seram Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics*, 22(14):e90-e98, 2006.

[25] Daniel H. Younger. Context-free language processing in time n^3. *Switching and Automata Theory, 1966., IEEE Conference Record of Seventh Annual Symposium on*, pages 7-20.

[26] B. Knudsen, and J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31:3423-3428, 2003.

[27] Robin Dowell and Sean Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(1):71, 2004.

[28] Das R, Baker D. Automated de novo prediction of native-like RNA tertiary structures. *Proc Natl Acad Sci*, 104(37):14664-14669, USA 2007.

[29] Sharma S, Ding F, Dokholyan NV. iFoldRNA: three-dimensional RNA structure prediction and folding. *Bioinformatics*, 24:1951–1952, 2008.

[30] Jonikas MA, Radmer RJ, Laederach A, Das R, Pearlman S, Herschlag D, Altman RB. Coarse grained modeling of large RNA molecules with knowledge-based potentials and structural filters. *RNA*, 15:189–199, 2009.

[31] Parisien M, Major F. The MC-Fold and MC-Sym pipeline infers RNA structure from sequence data. *Nature*, 452:51-55, 2008.

[32] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[33] H.L. Bodlaender and, T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[34] N. Robertson, and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Combin. Theory Ser. B*, 36(1):49–64, 1984.

[35] S. Arnborg, and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[36] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. *Computer Science Logic Lecture Notes in Computer Science* 533, pages 1–16, 1990.

[37] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability-a survey. *BIT*, 25(1):2–23, 1985.

[38] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for treedecomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[39] H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. *Automata, Languages and Programming Lecture Notes in Computer Science* , 317:405-118, 1988.

[40] Jon Kleinberg , Éva Tardos. Algorithm Design. Addison-Wesley, 2005.

[41] C.H. Papadimitriou. Computational Complexity. Addison-Wesley Publishing Company, inc.,1994.

[42] R. Nussinov and A. B. Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, 1980.

[43] J.T. Ngo and J. Marks. Computational complexity of a problem in molecular-structure prediction. *Protein Engineering*, 5(4):313–321, 1992.

[44] R. B. Lyngsø. Computational aspects of biological sequences and structures. Master's thesis, Department of Computer Science, University of Aarhus, June 1997.

[45] Dan Gusfield. Algorithms on Strings, Trees and Sequences: Computer Science and Computational biology. Cambridge University Press, 1997.

[46] Richard Durbin, Sean R. Eddy , Anders Krogh , Graeme Mitchison. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids . Cambridge University Press, 1998.

[47] M. Farley. Networks immune to isolated failures. *Networks,* 11(3) 255-268, 1981.

[48] Joseph A. Wald, Charles J. Colbourn. Steiner trees, partial 2–trees, and minimum IFI networks. *Networks,* 13(2) 159-167, 1983.

[49] Pooya Shareghi Arani. Graph Generating Systems for Predicting Biological Structures. Ph.d. Dissertation, Computer Science Dept., UGA, 2012.

[50] Michael Zuker and David Sanko. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology* , 46:591-621, 1984.

[51] Reinhard Diestel. Graph theory. Graduate Texts in Mathematics, volume 173. Springer-Verlag, 2nd edition, 2000.

[52] H.L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, (1993).

[53] Donald. J. Rose. On simple characterization of k-trees . *DISCRETE MATHEMATICS* 7(1974) 317-322 North-Holland Publishing company.

[54] Andreas Krause. Bounded Treewidth Graphs-A Survey. *German Russian Winter School St.Petesburg*, Russia , Technical University of Munich, 2003.

[55] H.L. Bodlaender . A Partial K-Arboretum of Graphs With Bounded Treewidth. *Theoretical Computer Science*, 209(1-2):1-45, 1998.

[56] R.G. Downey and M.R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109-131, 1995.

[57] Hopcroft, J.E., Motwani, R., and Ullman, J.D. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 3rd edition 2006.

[58] Salomaa, A. Jewels of Formal Language Theory, Computer Science Press, 1981.

[59] Lari, K. and Young, S.J. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35-56, 1990.

[60] Srebro, N. Maximum likelihood bounded tree-width Markov networks, *Artificial Intelligence*, 143:123–138, 2003.

[61] T. A. Sudkamp. Languages and Machines: An Introduction to the Theory of Computer Science. Addison-Wesley, 2005.

[62] J. Earley. An effcient context-free parsing algorithm. *Communications of the ACM*, 13(2), 1970.

[57] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346-1367,

2006.