

Parametric Problems on Graphs of Bounded Tree-width

TR 92-08
David Fernandez-Baca and Giora Slutzki

April 22, 1992

Iowa State University of Science and Technology
Department of Computer Science
226 Atanasoff
Ames, IA 50011

Parametric Problems on Graphs of Bounded Tree-width*

DAVID FERNÁNDEZ-BACA[†] AND GIORA SLUTZKI

Department of Computer Science, Iowa State University, Ames, IA 50011

April 8, 1992

Abstract

We consider optimization problems on weighted graphs where vertex and edge weights are polynomial functions of a parameter λ . We show that, if a problem satisfies certain regularity properties and the underlying graph has bounded tree-width, the number of changes in the optimum solution is polynomially bounded. We also show that the description of the sequence of optimum solutions can be constructed in polynomial time and that certain parametric search problems can be solved in $O(n \log n)$ time, where n is the number of vertices in the graph.

1 Introduction

We shall consider parametric optimization problems whose nonparametric version takes the following familiar form. Given a graph G with real-valued vertex and edge weight functions $w_V : V(G) \rightarrow \mathbf{R}$ and $w_E : E(G) \rightarrow \mathbf{R}$, respectively, find an optimum (i.e., minimum- or maximum-weight) subgraph H satisfying a property P . Well-known examples of such problems are minimum-weight dominating set, minimum-weight vertex cover, and the traveling salesman problem. Let us write $\mathbf{val}_G(H)$ to denote $\sum_{v \in V(H)} w_V(v) + \sum_{e \in E(H)} w_E(e)$, where H is a subgraph of G . We can express all optimum subgraph problems as

$$z_G^P = \text{opt}\{\mathbf{val}_G(H) : H \text{ a subgraph of } G \text{ satisfying } P\}, \quad (1)$$

where “opt” is either “min” or “max”, depending on the problem.

It is well known that many optimum subgraph problems that are NP-hard in general are polynomially solvable for restricted classes of graphs [GaJo79]. Recently, a long line of work has culminated in the development of various methodologies for devising polynomial-time (and, indeed, often *linear*-time) algorithms for graphs of *bounded tree-width* [AbFe92, ALS91, ArPr89, Bod87, BPT88, BLW87, Cou90, Wim87]

*A preliminary version of this paper will appear in the proceedings of the *3rd Scandinavian Conference on Algorithm Theory, 1992*.

[†]Supported in part by the National Science Foundation under grant No. CCR-8909626.

(for a definition of tree-width, see [RoSe86] or section 2 of this paper). While these approaches differ from each other in several respects, in essence they all deal with subgraph problems that have certain “regularity” properties that make them amenable to dynamic programming solutions. The class of regular problems is broad, and includes the subgraph problems mentioned above, as well as many others, such as the maximum cut problem and the Steiner tree problem (see, e.g., [ALS91, BPT88, BLW87]). Here we shall study the implications that regularity properties have on the parametric versions of these problems.

Parametric optimization problems arise in *sensitivity analysis* [Gus83], *minimum-ratio optimization* [Meg79, Meg83], *Lagrangian relaxation* [Fis81], and, in general, in environments where the data evolves continuously with time. We will focus on parametric optimum subgraph problems where vertex and edge weights are functions of a real-valued parameter λ . That is, we are given a graph G with vertex and edge weight functions $W_V : V(G) \times \mathbf{R} \rightarrow \mathbf{R}$ and $W_E : E(G) \times \mathbf{R} \rightarrow \mathbf{R}$, respectively. Let us write $\text{Val}_G(H, \lambda)$ to denote $\sum_{v \in V(H)} W_V(v, \lambda) + \sum_{e \in E(H)} W_E(e, \lambda)$, where H is a subgraph of G . The function of interest to us is

$$Z_G^P(\lambda) = \text{opt}\{\text{Val}_G(H, \lambda) : H \text{ a subgraph of } G \text{ satisfying } P\}. \quad (2)$$

In what follows, for concreteness, we shall often deal specifically with minimization problems. All the results we shall derive easily extend to maximization problems. In most of our subsequent discussions we will fix P , while G may vary. Thus, we shall often write Z_G instead of Z_G^P . Similarly, we shall often write z_G instead of z_G^P .

For tractability, we shall consider the case where weights are polynomial functions of λ . Thus, for any subgraph H of G , $\text{Val}_G(H, \lambda)$ is a polynomial in λ . By equation (2), Z_G is the lower envelope of the set of polynomials $\text{Val}_G(H, \lambda)$ such that H is a subgraph of G satisfying P . Thus, Z_G is a piecewise-polynomial function of λ . Z_G partitions the λ -axis into a sequence of intervals, where each interval is the maximal set of λ -values for which $Z_G(\lambda) = \text{Val}_G(H, \lambda)$ for some particular subgraph H satisfying P . The boundary points between intervals are called *breakpoints*. We can represent Z_G by (1) listing these intervals in order, from left to right and (2) for each interval providing the associated optimum subgraph H . Clearly, such a representation is finite. In the special case where weights are linear functions of λ , Z_G will be a piecewise-linear concave function [Gus80] (Z_G is convex for maximization problems).

We shall concentrate on two kinds of issues: construction and search. In construction problems we shall be interested in computing the entire representation of the function Z_G . Search problems involve finding a value of λ at which a particular event occurs. Examples are the problems of finding the next breakpoint of Z_G or a value of λ that maximizes Z_G . These problems and their applications are discussed further in sections 4, 5, and 6.

There are two main results in the paper. Both deal with *regular* optimum subgraph problems (in the sense of Bern et al. [BLW87] and Borie et al. [BPT88]) on graphs of bounded tree-width. First, we show that for every regular graph property P , the number of breakpoints of Z_G^P is polynomially-bounded in $n = |V(G)|$. As a byproduct of the proof we obtain a polynomial-time algorithm to construct Z_G^P . The second result is a proof that, for every regular property P , there exist $O(n \log n)$ algorithms for certain kinds of parametric search problems, including the two search problems stated above.

Related Work. Several researchers have considered parametric versions of combinatorial optimization problems. Murty [Mur80] showed that the number of break-points for parametric linear programming problems can be exponential in the number of variables. Subsequently, Carstensen [Car83] proved exponential lower bounds for, among other problems, parametric minimum cut and knapsack. Van Hoesel *et al.* [vHKRW89] have compiled an extensive bibliography on parametric computing. The algorithmic approach followed in this paper was first used in [FeSl89] to analyze various optimization problems on trees, including weighted vertex cover and dominating set. It has subsequently been used to analyze the parametric maximum independent set problem on outerplanar graphs [ZhGo91] and parametric nonserial dynamic programming problems (and, as a special case, independent set) on partial k -trees [FeMe90]. Many of the ideas used here are based on the work of Megiddo; specifically, on [Meg79, Meg83]. Search and construction algorithms for an important special case of the parametric maximum flow problem are presented in [GGT89]. A theory of converting nonparametric algorithms into parametric ones is presented by Eaves and Rothblum in [EaRo89]. A field related to parametric optimization is dynamic computational geometry, originally studied by Atallah [Ata85], which deals with geometric problems where points move in space following prescribed trajectories.

Organization of the paper. Section 2 reviews the notions of tree-width and regularity. In section 3 we present some results relating parse trees and tree-decompositions and review the dynamic programming approach to solving optimum subgraph problems. In section 4 we study the properties of Z_G^P . In section 5 we present our parametric search algorithms. Finally, section 6 discusses related results and open problems.

2 Preliminaries

2.1 Tree-width

The following definition is due to Robertson and Seymour [RoSe86].

Definition 1 Let G be an undirected graph. A *tree decomposition* of G is a labeled tree (T, χ) , where χ is the labeling function for T , such that for all $i \in V(T)$, $\chi(i) = \chi_i \subseteq V(G)$, and such that the following conditions hold.

1. $\bigcup_{i \in V(T)} \chi_i = V(G)$.
2. For every $(u, v) \in E(G)$, $\{v, u\} \subseteq \chi_i$ for some $i \in V(T)$.
3. If j lies on the path of T from i to k , then $\chi_i \cap \chi_k \subseteq \chi_j$.

The *width* of a tree-decomposition is $\max_{i \in V(T)} (|\chi_i| - 1)$. The *tree-width* of a graph G is the minimum over all tree-decompositions (T, χ) of G of the width of (T, χ) .

We shall write Γ_w to denote the set of all graphs of tree-width at most w . Many important classes of graphs have bounded tree-width, including trees, series-parallel graphs, bandwidth- k graphs, k -outerplanar graphs, and partial k -trees [ArPr89, Bod88, vLe90].

It is known that a graph G is a partial k -tree if and only if $G \in \Gamma_k$ [vLe90]. The graphs in Γ_w have a useful separator theorem. We say that sets $Y, Y' \subseteq V(G)$ are *separated* by $S \subseteq V(G)$ if every path in G from Y to Y' goes through S . A set $S \subseteq G$ is a *separator* of G if $G - S$ is not connected. The next result follows easily from Theorem 2.5 of [RoSe86].

Theorem 2.1 *Let $G \in \Gamma_w$ and suppose $Q \subseteq V(G)$. Then, there exists a partition (B_1, B_2, B_3, S) of $V(G)$ with $|S| \leq w + 1$ such that for $i, j \in \{1, 2, 3\}$, $i \neq j$, B_i and B_j are separated by S , and $|B_i \cap Q| \leq |Q - S|/2$ for $i = 1, 2, 3$.*

The partition (B_1, B_2, B_3, S) whose existence is claimed in Theorem 2.1 can be computed in polynomial time by exhaustive enumeration [RoSe86]. Linear-time separator algorithms exist for graphs of tree-width at most 3, while *approximate* separators (see section 6) for graphs of higher tree-width can be found efficiently. We shall return to this issue in section 6.

2.2 Regular Graph Properties

The various subgraph problems that are amenable to dynamic programming on graphs of bounded tree-width share two key properties. First, the space of potential solutions to these problems can be partitioned into a finite number of equivalence classes [ALS91, ArPr89, Bod87, BPT88, BLW87, Cou90]. Second, there is a finite set of rules whereby partial solutions, computed for portions of the input graph, can be combined into larger partial solutions. Rules are expressed in tables which are fixed for each problem and each family of graphs. These two facts are essential in the proof of our results and, for this reason, we shall explain them in some detail. For convenience, we shall follow the terminology of [BLW87, BPT88], as it seems to provide a rather explicit view of the solution mechanism.

Let \mathcal{G} be the set of all finite graphs. A graph $G \in \mathcal{G}$ is a *k -terminal graph* if it is given together with a list $\mathbf{terms}(G) = \langle t_1, \dots, t_s \rangle$, $1 \leq s \leq k$, of distinct vertices of G called *terminals*. The set of all k -terminal graphs will be denoted by \mathcal{G}_k . A *k -terminal graph composition operator* is a (partial) function $\varphi : \mathcal{G}_k^r \rightarrow \mathcal{G}_k$, where $r = r(\varphi) \geq 0$ is the *arity* of φ . The resulting graph $G = \varphi(G_1, \dots, G_r) \in \mathcal{G}_k$ is obtained by identifying the terminals of G_1, \dots, G_r in some precisely prescribed way. The terminals of G are obtained from the terminals of the composing graphs. See [Wim87, BPT88] for two ways to formalize this concept. We should note at this point that in weighted graph problems (parametric or not), when two nodes get identified by an operation φ , these nodes are always assumed to have the same weight (or weight function).

In the remainder of this section, \mathcal{R} will denote a finite set of k -terminal composition operations and $\mathcal{R}_i \subseteq \mathcal{R}$ is the subset of operations of arity i . Note that \mathcal{R}_0 is a subset of \mathcal{G}_k . The graphs in \mathcal{R}_0 are called the *base* or *primitive* graphs. We will always assume that $\mathcal{R}_0 \neq \emptyset$.

A *family of decomposable graphs over \mathcal{R}* is the smallest family of graphs $\mathcal{R}^* \subseteq \mathcal{G}_k$ defined recursively as follows:

- (D1) $\mathcal{R}_0 \subseteq \mathcal{R}^*$.
- (D2) For all $r \geq 1$, for each $\varphi \in \mathcal{R}_r$, and for all $G_1, \dots, G_r \in \mathcal{R}^*$, if $\varphi(G_1, \dots, G_r)$ is defined, then $\varphi(G_1, \dots, G_r) \in \mathcal{R}^*$.

The equality $G = \varphi(G_1, \dots, G_r)$ is called a *decomposition of G with respect to \mathcal{R}* . Let (T, δ) be a rooted, ordered tree with labeling function $\delta : V(T) \rightarrow \mathcal{R}$ satisfying the following compatibility requirement: If $v \in V(T)$ has $r \geq 0$ children in T , then $\delta(v) \in \mathcal{R}_r$. Given such a tree (T, δ) , we define an induced partial function $\gamma_T : V(T) \rightarrow \mathcal{R}^*$ as follows:

(R1) If $v \in V(T)$ is a leaf (i.e., $\delta(v) \in \mathcal{R}_0$), then $\gamma_T(v) = \delta(v)$.

(R2) If $v \in V(T)$ is an internal node with children v_1, \dots, v_r , then

$$\gamma_T(v) = \delta(v)(\gamma_T(v_1), \dots, \gamma_T(v_r)).$$

We define the set $\mathcal{T}_{\mathcal{R}}$ of *trees over \mathcal{R}* as the set of all trees (T, δ) for which the induced function γ_T is total. Let $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$. For $v \in V(T)$, $\gamma_T(v) \in \mathcal{R}^*$ is the graph *represented by* the subtree of T rooted at v . In particular, if v is the root of T , then $\gamma_T(v)$ is the *graph represented by T* . A tree $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$ is a *parse tree* of $G \in \mathcal{R}^*$ if $G = \gamma_T(v)$, where v is the root of T . Obviously, every $G \in \mathcal{R}^*$ has at least one parse tree $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$, and every $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$ is a parse tree of a unique $G \in \mathcal{R}^*$.

When dealing with optimization problems on graphs, we will be interested in the set of all graph \times subgraph pairs $\hat{\mathcal{G}} = \{(G, H) : G \in \mathcal{G}, H \text{ a subgraph of } G\}$ and its k -terminal version, $\hat{\mathcal{G}}_k$. For $D = (G, H) \in \hat{\mathcal{G}}_k$, the *signature* of D is the list $\mathbf{terms}(G)$, together with information as to which nodes in this list are in H . We extend the k -terminal operations $\varphi \in \mathcal{R}$ to $\hat{\mathcal{G}}_k$ by letting $\hat{\mathcal{R}} = \bigcup_{r \geq 0} \hat{\mathcal{R}}_r$, where

$$\hat{\mathcal{R}}_0 = \{(\varphi, H) : \varphi \in \mathcal{R}_0, H \text{ a subgraph of } \varphi\}$$

and, for $r \geq 1$,

$$\hat{\mathcal{R}}_r = \{\hat{\varphi} : \varphi \in \mathcal{R}_r\}$$

with $\hat{\varphi}$ defined as follows. If $\varphi(G_1, \dots, G_r)$ is defined, then

$$\hat{\varphi}((G_1, H_1), \dots, (G_r, H_r)) = (G, H)$$

where $G = \varphi(G_1, \dots, G_r)$ and H is specified by $V(H) = \bigcup_{i=1}^r V(H_i)$, and $E(H) = \bigcup_{i=1}^r E(H_i)$, modulo vertex identification resulting from application of φ . Analogously with the definition of \mathcal{R}^* , we can define a family $\hat{\mathcal{R}}^* \subseteq \hat{\mathcal{G}}_k$ as the closure of the set $\hat{\mathcal{R}}$.

Of great importance here will be a special kind of predicates on $\hat{\mathcal{G}}$, called *regular predicates*. We shall define them in an algebraic framework. Let $\mathcal{D} = \hat{\mathcal{R}}^*$. Then $\mathcal{D} = (\mathcal{D}, \hat{\mathcal{R}})$ is an algebra, with domain (carrier) \mathcal{D} and operations $\hat{\mathcal{R}}$. Suppose $\mathcal{C} = (\mathcal{C}, \mathcal{Q})$ is an algebra such that there is an arity-preserving one-to-one correspondence between $\hat{\mathcal{R}}$ and \mathcal{Q} — i.e. \mathcal{C} is *similar* to \mathcal{D} . Let P be a predicate on $\hat{\mathcal{G}}$ and let $h : \mathcal{D} \rightarrow \mathcal{C}$ be a mapping. We say that h *respects* P if for every $D_1, D_2 \in \mathcal{D}$,

$$(H1) \quad h(D_1) = h(D_2) \implies P(D_1) = P(D_2).$$

We note that the D_i 's are in $\hat{\mathcal{G}}_k$, while P is a predicate on $\hat{\mathcal{G}}$. When we write “ $P(D_i)$,” we intend that the value of P for D_i should not depend on the terminals associated with D_i . The mapping h is a *homomorphism with respect to \mathcal{D}* (or \mathcal{R}) if for every $\hat{\varphi} \in \hat{\mathcal{R}}$, $r = r(\hat{\varphi})$, and for every $D_1, \dots, D_r \in \mathcal{D}$ such that $\hat{\varphi}(D_1, \dots, D_r)$ is defined,

$$(H2) \quad h(\hat{\varphi}(D_1, \dots, D_r)) = \tilde{\varphi}(h(D_1), \dots, h(D_r))$$

where $\tilde{\varphi} \in \mathcal{Q}$ is the unique operation corresponding to $\hat{\varphi} \in \hat{\mathcal{R}}$.

We say that a predicate P is *regular with respect to \mathcal{D} (or \mathcal{R})* if there exists a finite algebra \mathcal{C} similar to \mathcal{D} and a mapping $h : \mathcal{D} \rightarrow \mathcal{C}$ that satisfies (H1) and (H2). Note that h defines an equivalence relation \sim_h on \mathcal{D} , where $D_1 \sim_h D_2$ if and only if $h(D_1) = h(D_2)$, and that \sim_h has at most $|\mathcal{C}|$ equivalence classes.

From now on, rather than referring to the algebra \mathcal{C} (which, in a sense, is exterior to \mathcal{D}), we will let $\mathcal{C} = \{C_1, \dots, C_N\}$ be the set of equivalence classes of \mathcal{D} with respect to \sim_h (i.e., it is the *quotient* \mathcal{D}/\sim_h). Every $\hat{\varphi} \in \hat{\mathcal{R}}_r$ (for all $r \geq 0$) is thus “lifted” to an operation $\tilde{\varphi} : \mathcal{C}^r \rightarrow \mathcal{C}$ defined by

$$\tilde{\varphi}([D_1], \dots, [D_r]) = [\tilde{\varphi}(D_1, \dots, D_r)],$$

where $[D_i]$ is the equivalence class containing D_i . These definitions are correct because \sim_h is actually a *congruence* relation on \mathcal{D} . Without loss of generality, we will assume that each equivalence class uniquely determines the signature of its elements; i.e., for every i ($1 \leq i \leq N$), every $D_1, D_2 \in C_i$ have the same signature. If the C_i 's do not have this property, then they can be refined to achieve it. We will refer to \mathcal{C} as the *set of equivalence classes of \mathcal{D} with respect to predicate P* . A class C_i is said to be *accepting* if there exists a pair $D \in C_i$ such that $P(D)$ holds. We note that several well-known optimum subgraph problems have been shown to be regular on *any* family of decomposable graphs. These problems include dominating set, maximum cut, Steiner tree, traveling salesman, and independent set [BPT88].

We will need one further piece of notation. For $\varphi \in \mathcal{R}_r$, $\text{MT}(\varphi, i)$ will denote the set of all ordered r -tuples (i_1, \dots, i_r) such that $\tilde{\varphi}(C_{i_1}, \dots, C_{i_r}) = C_i$.

3 The Basic Algorithm

In this section we shall describe the scheme underlying our subsequent results. We first show the existence of parse trees with certain useful properties for every $G \in \Gamma_w$. Next, as a prelude to the discussion of parametric problems in sections 4 and 5, we review the dynamic programming algorithm for nonparametric optimum subgraph problems of Bern et al. [BLW87].

3.1 Separators and Parse Trees

The following theorem, due to Wimer [Wim87], shows the close relationship between graphs of bounded tree-width and decomposable graphs:

Theorem 3.1 *There exists a finite family \mathcal{R} of $(w + 1)$ -terminal graph composition operators such that $\Gamma_w = \mathcal{R}^*$.*

For our purposes, we shall need a variant of this result. Let \mathcal{R} be a family of k -terminal composition operations. A decomposition $G = \varphi(G_1, \dots, G_r)$ with respect to \mathcal{R} is said to be *balanced* if $|V(G_i)| \leq |V(G)|/2 + c$, where c is a constant that depends only on \mathcal{R} . A parse tree (T, δ) of an n -vertex graph G is said to be *balanced* if

(B1) $|V(T)| = O(n)$,

(B2) the height of T is $O(\log n)$, and

(B3) for every internal node v of T with children v_1, \dots, v_r , the decomposition $\gamma_T(v) = \delta(v)(\gamma_T(v_1), \dots, \gamma_T(v_r))$ is balanced with respect to \mathcal{R} .

Finally, we shall write $\mathcal{U}[k, r]$ to denote the set of all k -terminal graph composition operators $\varphi : \mathcal{G}_k^s \rightarrow \mathcal{G}_k$ where $0 \leq s \leq r$, and where, for each φ of arity zero, $|V(\varphi)| \leq k$. Note that for every fixed k and r the size of $\mathcal{U}[k, r]$ is bounded by a constant.

The proof of the next result relies on a simple observation. Let G be a graph and, for $X \subseteq V(G)$, let $G[X]$ denote the subgraph of G induced by X . Suppose (A_1, \dots, A_l, S) is a partition of $V(G)$ such that, for $i \neq j$, A_i and A_j are separated by S . Then, one can view G as the composition of $|S|$ -terminal graphs G_1, \dots, G_l , where $G_i = G[A_i \cup S]$ and $\mathbf{terms}(G_i)$ consists of the vertices of S in some arbitrary (but fixed) order. Obviously, the corresponding composition operator, say φ , is an element of $\mathcal{U}[|S|, l]$.

Theorem 3.2 *For each fixed w , every $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$ has a balanced decomposition with respect to $\mathcal{U}[4w+4, 5]$.*

Proof. We shall argue that the following procedure, based on ideas from [Lag91], produces the desired decomposition for every $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$. We write $\mathbf{terms}(G)$ to denote the set of vertices in $\mathbf{terms}(G)$.

Procedure DECOMPOSE

Input: $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$, where $\mathbf{terms}(G) = \langle t_1, \dots, t_k \rangle$.

Output: Balanced decomposition $G = \varphi(G_1, \dots, G_r)$, with $\varphi \in \mathcal{U}[4w+4, 5]$.

Step 1. If $|V(G)| \leq 4w+4$, return the decomposition $G = G$.

Step 2. Find a partition (A_1, A_2, A_3, S_1) of $V(G)$ satisfying Theorem 2.1 with $Q = V(G)$. Let $H_i = G[A_i]$, for $1 \leq i \leq 3$. Assume w.l.o.g. that $|A_1 \cap \mathbf{terms}(G)| \geq |A_2 \cap \mathbf{terms}(G)| \geq |A_3 \cap \mathbf{terms}(G)|$.

Step 3. If $|A_1 \cap \mathbf{terms}(G)| \leq k/2$, return the decomposition $G = \varphi(G_1, G_2, G_3)$, where, for $1 \leq i \leq 3$, $G_i = G[A_i \cup S_1]$ and $\mathbf{terms}(G_i)$ consists of S_1 , together with the vertices in $\mathbf{terms}(G) \cap A_i$, in any order, and φ is an appropriate composition operator in $\mathcal{U}[4w+4, 5]$.

Step 4. If $|A_1 \cap \mathbf{terms}(G)| > k/2$, find a partition (B_1, B_2, B_3, S_2) of H_1 satisfying Theorem 2.1 with $Q = \mathbf{terms}(G) \cap A_1$. For $i = 1, 2, 3$, let $G_i = G[B_i \cup S_1 \cup S_2]$ and let $\mathbf{terms}(G_i)$ consist of the vertices in S_1, S_2 , and $\mathbf{terms}(G) \cap B_i$, in any order. For $i = 4, 5$, let $G_i = G[A_{i-2}]$ and let $\mathbf{terms}(G_i)$ consist of S_1 , together with the vertices in $\mathbf{terms}(G) \cap A_{i-2}$, in any order. Return the decomposition $G = \varphi(G_1, \dots, G_5)$, where φ is an appropriate composition operator in $\mathcal{U}[4w+4, 5]$.

If $|V(G)| \leq 4w + 4$, then G is an operator of arity zero in $\mathcal{U}[4w + 4, 5]$. Hence, the output returned by Step 1 is a balanced decomposition of G with respect to $\mathcal{U}[4w + 4, 5]$. Now consider the output returned by Steps 3 and 4. In either case, it is easy to see that the use of Theorem 2.1 ensures that each G_i in the output decomposition satisfies $|V(G_i)| \leq |V(G)|/2 + w + 1$ and $G_i \in \Gamma_w$. All that is left to show is that each such G_i is in \mathcal{G}_{4w+4} . Recall that, by Theorem 2.1, $|S_1|, |S_2| \leq w + 1$. If the decomposition is returned by Step 3, then $|\text{terms}(G_i)| = |S_1| + |\text{terms}(G) \cap A_i|$. But, $|\text{terms}(G) \cap A_i| \leq k/2 \leq 2w + 2$, so $|\text{terms}(G_i)| \leq 3w + 3$ and, thus $G_i \in \mathcal{G}_{4w+4}$. If the decomposition is returned by Step 4, then, by the same reasoning, $|\text{terms}(G_i)| \leq 3w + 3$ for $i = 4, 5$. For $i = 1, 2, 3$, $|\text{terms}(G_i)| = |S_1| + |S_2| + |\text{terms}(G) \cap B_i|$. But $|\text{terms}(G) \cap B_i| \leq k/2 \leq 2w + 2$, so $|\text{terms}(G_i)| \leq 4w + 4$. Thus, again, for every G_i in the decomposition, $G_i \in \mathcal{G}_{4w+4}$. \square

Corollary 3.3 *For every fixed w there exists $\mathcal{R} \subseteq \mathcal{U}[4w+4, 5]$ such that (i) $\Gamma_w \subseteq \mathcal{R}^*$, and (ii) every $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$, has a balanced parse tree (T, δ) in $\mathcal{T}_{\mathcal{R}}$.*

Proof. A balanced parse tree (T, δ) of any $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$ can be produced as follows. If $n = |V(G)| \leq 4w + 4$, (T, δ) consists of a single node v where $\delta(v) = G$. Otherwise, use procedure DECOMPOSE to find a balanced decomposition $G = \varphi(G_1, \dots, G_r)$. Next, recursively construct a parse (T_i, δ_i) for each G_i . The parse tree of G will consist of a root v where $\delta(v) = \varphi$, and subtrees $(T_1, \delta_1), \dots, (T_r, \delta_r)$. It is not hard to show that (T, δ) has $O(n)$ vertices and $O(\log n)$ height (see [Lag91] for a similar construction). Obviously, for every $v \in V(T)$, $\delta(v) \in \mathcal{U}[4w + 4, 5]$. \square

Let $w \geq 1$ be an integer. We shall say that a family \mathcal{R} of composition operators is *w-adequate* if $\mathcal{R} \subseteq \mathcal{U}[4w + 4, 5]$ and every $G \in \mathcal{G}_{4w+4} \cap \Gamma_w$ has a balanced parse tree in $\mathcal{T}_{\mathcal{R}}$, which can be obtained as explained in Corollary 3.3. It follows from the proof of Corollary 3.3 that there exists a *w-adequate* family of composition operators for every $w \geq 1$.

3.2 Dynamic Programming on Decomposable Graphs

The following result was proved in [BLW87].

Theorem 3.4 *Suppose that $\mathcal{F} = \mathcal{R}^*$ is a class of decomposable graphs and P is a property that is regular with respect to \mathcal{R} . Then there exists a linear-time algorithm that, given a linear-size parse tree of $G \in \mathcal{F}$, finds an optimum-weight subgraph of G satisfying P .*

We sketch the proof of this theorem, because the underlying approach serves as a basis for our subsequent results. Let $\mathcal{C} = \{C_1, \dots, C_N\}$ denote the set of equivalence classes of $\hat{\mathcal{R}}^*$ with respect to P . Let G be in \mathcal{F} . We define $z_G^{(i)}$ and z_G to be

$$z_G^{(i)} = \min \{ \{+\infty\} \cup \{ \text{val}_G(H) : (G, H) \in C_i \} \} \quad (3)$$

and

$$z_G = \min \{ \{+\infty\} \cup \{ z_G^{(i)} : C_i \text{ is an accepting class} \} \}. \quad (4)$$

Note that the value of z_G is equal to $\text{val}_G(H)$, where H is an optimum subgraph of G satisfying P (in the sense of equation (1)). If G is a primitive graph, then $z_G^{(i)}$ can be computed directly from equation (3) — i.e., by exhaustive enumeration. Otherwise, suppose $G = \varphi(G_1, \dots, G_r)$ is a decomposition of G with respect to \mathcal{R} . Then, $z_G^{(i)}$ can be computed using the following equation

$$z_G^{(i)} = \min \left\{ \sum_{j=1}^r z_{G_j}^{(i_j)} - \text{sv}[\varphi; \mathbf{i}] : \mathbf{i} = (i_1, \dots, i_r) \in \text{MT}(\varphi, i) \right\}, \quad (5)$$

where $\text{sv}[\varphi; \mathbf{i}]$ is the sum of the weights of shared vertices; i.e., those vertices that have been accounted for more than once in $\sum_{j=1}^r z_{G_j}^{(i_j)}$. Observe that, given the $z_{G_j}^{(i_j)}$'s, equation (5) can be evaluated in $O(1)$ time. Clearly, there may exist other decompositions of G with respect to \mathcal{R} . Even though each of these decompositions will lead to a different equation of the form (5), in all cases, the value of $z_G^{(i)}$ will be the same. As argued in [BLW87], these facts and the existence of linear-size parse trees can be used to obtain linear-time algorithms to compute z_G .

4 Parametric Problems

We shall now study the properties of Z_G^P when P is a regular graph property and G has bounded tree-width. We use the following notation. Let d be a nonnegative integer. The term *d-th degree polynomial* will be used to refer to any polynomial of degree at most d . A function $f : \mathbf{R} \rightarrow \mathbf{R}$ is a *d-th degree piecewise polynomial function (d-ppf)* if it is the lower envelope of some finite set of d -th degree polynomials in λ ; we write $b(f)$ to denote the number of breakpoints of f . The following result gives upper bounds for the number of breakpoints of the sum and lower envelope of d -ppfs. It is a generalization of a similar result for linear functions proved in [FeSl89].

Lemma 4.1 *Let f_1, \dots, f_m be d -ppf's and let $g_1 = \sum_{j=1}^m f_j$ and $g_2 = \min_{1 \leq j \leq m} f_j$. Then*

- (i) $b(g_1) \leq \sum_{j=1}^m b(f_j)$, and
- (ii) $b(g_2) \leq s(m, d) \left(\sum_{j=1}^m b(f_j) + 1 \right) - 1$,

where $s(m, d)$ is a function that depends only on m and d .

Proof. Observe that the breakpoints of the f_j 's partition the λ -axis into at most $\sum_{j=1}^m b(f_j) + 1$ intervals. For part (i), note that, within each interval, g_1 is a d -th degree polynomial, since it is the sum of m d -th degree polynomials. Thus, g_1 has no breakpoints in the interior of any of these intervals and, consequently, g_1 has at most $\sum_{j=1}^m b(f_j)$ breakpoints.

For part (ii), note that, within each interval I , g_2 is the lower envelope of m d -th degree polynomials f'_1, \dots, f'_m . Since any pair of these functions can intersect at most

d times, g_2 has at most $dm(m-1)/2$ breakpoints in the interior of I . Therefore,

$$\begin{aligned} b(g_2) &\leq \frac{dm(m-1)}{2} \left(\sum_{j=1}^m b(f_j) + 1 \right) + \sum_{j=1}^m b(f_j) \\ &\leq s(m, d) \left(\sum_{j=1}^m b(f_j) + 1 \right) - 1, \end{aligned}$$

where $s(m, d) = d(m^2 - m)/2 + 1$. \square

Remark. Since g_2 is the lower envelope of at most $M = \sum_{j=1}^m b(f_j) + m$ functions, we can use well-known results on Davenport-Schinzel sequences [Sha87] to prove that for $d = 1$ and $d = 2$, $b(g_2) \leq M$ and $b(g_2) \leq 2M - 1$, respectively. Since both bounds are better than those provided by Lemma 4.1, it is tempting to use the same strategy for larger values of d . This will not suffice for our purposes, however, because for $d \geq 3$ and fixed m , the lower envelope of a set of M d -th degree polynomials will, in general, have a number of breakpoints that is superlinear (although only slightly so) in M [Sha87].

The main result of this section is the following theorem. In its proof, we assume a model of computation where finding the roots of a d -th degree polynomial function is a primitive operation.

Theorem 4.2 *Let $w \geq 1$ and let P be a predicate that is regular with respect to some w -adequate set of composition operators \mathcal{R} . Then, for any $G \in \Gamma_w$ whose vertex and edge weights are d -th degree polynomial functions of λ , $b(Z_G^P)$ is polynomially bounded in $|V(G)|$. Furthermore, given a linear-size parse tree (T, δ) of G , Z_G^P can be constructed in polynomial time.*

Proof. The main idea behind our proof is *algorithm simulation*. This technique, which is inspired, in part, by [Meg79], was used in [FeSl89]. Let us refer to the dynamic programming algorithm described in section 3.2 as *algorithm \mathcal{A}* . Note that if we fix λ and make $w_V(v) = W_V(v, \lambda)$ for every $v \in V(G)$ and $w_E(e) = W_E(e, \lambda)$ for every $e \in E(G)$, then $z_G = Z_G(\lambda)$. Thus, we can use \mathcal{A} to compute $Z_G(\lambda)$ for *any* fixed λ . Based on this observation, we shall prove Theorem 4.2 by analyzing an algorithm \mathcal{A}_C , derived from \mathcal{A} , that constructs Z_G in its entirety.

Algorithm \mathcal{A}_C simulates the behavior of algorithm \mathcal{A} for all possible parameter values at once. That is, \mathcal{A}_C manipulates d -ppf's instead of real numbers. Notice that algorithm \mathcal{A} carries out only comparisons, additions, and subtractions. Wherever algorithm \mathcal{A} adds or subtracts real numbers, \mathcal{A}_C adds or subtracts d -ppf's and wherever algorithm \mathcal{A} finds the maximum or minimum of real numbers, \mathcal{A}_C computes upper or lower envelopes of d -ppf's. In order to specify algorithm \mathcal{A}_C more precisely, we define a function $Z_G^{(i)} : \mathbf{R} \rightarrow \mathbf{R}$ that is the parametric analog of $z_G^{(i)}$, for $1 \leq i \leq N$:

$$Z_G^{(i)}(\lambda) = \min \{ \{+\infty\} \cup \{ \text{Val}_G(H, \lambda) : (G, H) \in C_i \} \}, \quad (6)$$

and a function $Z_G : \mathbf{R} \rightarrow \mathbf{R}$ that is the parametric analog of z_G :

$$Z_G(\lambda) = \min \left\{ \{+\infty\} \cup \{Z_G^{(i)}(\lambda) : C_i \text{ is an accepting class}\} \right\}. \quad (7)$$

Note that Z_G and $Z_G^{(i)}$ are d -ppf's.

Algorithm \mathcal{A}_C proceeds as follows. If G is a primitive graph, then, for $i = 1, \dots, N$, \mathcal{A}_C computes $Z_G^{(i)}$ directly from its definition (equation (6)) by taking the the lower envelope of all polynomials $\text{Val}_G(H, \lambda)$, such that $(G, H) \in C_i$. Since \mathcal{R}_0 is a finite set of finite graphs, there exists a constant c_0 such that for every i , $c_0 \geq |\{(G', H') : G' \in \mathcal{R}_0 \text{ and } (G', H') \in C_i\}|$. Thus, $Z_G^{(i)}$ can be constructed in $O(1)$ time and $b(Z_G^{(i)}) = O(1)$.

If G is not a primitive graph, algorithm \mathcal{A}_C computes $Z_G^{(i)}$ via a counterpart to equation (5). Suppose $G = \varphi(G_1, \dots, G_r)$ is a decomposition of G with respect to \mathcal{R} . Then,

$$Z_G^{(i)}(\lambda) = \min \left\{ \sum_{j=1}^r Z_{G_j}^{(i_j)}(\lambda) - \text{SV}[\varphi; \mathbf{i}](\lambda) : \mathbf{i} = (i_1, \dots, i_r) \in \text{MT}(\varphi, i) \right\}, \quad (8)$$

where, as in equation (5), $\text{SV}[\varphi; \mathbf{i}]$ is the sum of the weight functions of the vertices that contribute more than once to $\sum_{j=1}^r Z_{G_j}^{(i_j)}(\lambda)$. Note that, because all graph \times subgraph pairs in an equivalence class have the same signature, the function $\text{SV}[\varphi; \mathbf{i}]$ is a d -th degree polynomial for all φ and \mathbf{i} . Thus, we have expressed $Z_G^{(i)}$ as the lower envelope of certain functions that are sums of d -ppf's. If these d -ppf's have a polynomial number of breakpoints, then Lemma 4.1 implies that so will $Z_G^{(i)}$; moreover, it follows that $Z_G^{(i)}$ is computable in polynomial time. These observations will form the basis of our proof. Let us define

$$\beta(n) = \max\{b(Z_G) : G \in \Gamma_w, |V(G)| \leq n\}$$

and

$$\beta^{(i)}(n) = \max\{b(Z_G^{(i)}) : G \in \Gamma_w, |V(G)| \leq n\}.$$

By equation (7) and Lemma 4.1, part (ii), we have that

$$\begin{aligned} b(Z_G) &\leq s(N_A, d) \sum \{b(Z_G^{(i)}) : C_i \text{ is an accepting class}\} + s(N_A, d) - 1, \\ &\leq c_1 \max_G \left\{ \sum \{b(Z_G^{(i)}) : C_i \text{ is an accepting class}\} \right\} + c_1 - 1, \\ &\leq c_1 \sum \{\beta^{(i)}(n) : C_i \text{ is an accepting class}\} + c_1 - 1, \end{aligned}$$

where N_A denotes the number of accepting classes, $c_1 = s(N_A, d)$, and the maximum is taken over all n -vertex graphs $G \in \Gamma_w$. Maximizing on the left-hand side over all n -vertex graphs $G \in \Gamma_w$, we have

$$\beta(n) \leq c_1 \sum \{\beta^{(i)}(n) : C_i \text{ is an accepting class}\} + c_1 - 1. \quad (9)$$

Since N_A and d are constants, so is c_1 , and polynomial bounds on the $\beta^{(i)}(n)$'s imply polynomial bounds on $\beta(n)$.

As observed above, for $G \in \mathcal{R}_0$, $b(Z_G^{(i)}) = O(1)$. Thus, there exists a constant c_2 such that, if $n_0 = \max\{|V(G)| : G \in \mathcal{R}_0\}$, then $\beta^{(i)}(n) \leq c_2$ for all $n \leq n_0$ and all

i. For $n > n_0$, we can use equation (8). Note that this equation provides us with different ways to compute $Z_G^{(i)}$, depending on which decomposition of G we use. For an arbitrary decomposition $G = \varphi(G_1, \dots, G_r)$, we have, using Lemma 4.1,

$$\begin{aligned} b(Z_G^{(i)}) &= b\left(\min\left\{\sum_{j=1}^r Z_{G_j}^{(i_j)} - \text{SV}[\varphi; \mathbf{i}] : \mathbf{i} = (i_1, \dots, i_r) \in \text{MT}(\varphi, i)\right\}\right) \\ &\leq c_3 \sum\left(\sum_{j=1}^r b(Z_{G_j}^{(i_j)}) : (i_1, \dots, i_r) \in \text{MT}(\varphi, i)\right) + c_3 - 1 \end{aligned} \quad (10)$$

where $c_3 = s(\max_{\varphi, i} |\text{MT}(\varphi, i)|, d)$. Note again that the term $\text{SV}[\varphi; \mathbf{i}]$ is a d -th degree polynomial in λ and, hence, does not contribute to equation (10). Note also that, while the value of $b(Z_G^{(i)})$ is independent of the decomposition used to compute $Z_G^{(i)}$, the right-hand side of equation (10) does, in general, depend on this decomposition. Since \mathcal{R} is w -adequate, G has at least one balanced decomposition. Since equation (10) holds for every decomposition of G , there is some constant α , $0 < \alpha < 1$,

$$\begin{aligned} b(Z_G^{(i)}) &\leq c_3 \sum\left\{\sum_{j=1}^r \beta^{(i_j)}(\alpha|V(G)|) : (i_1, \dots, i_r) \in \text{MT}(\varphi, i)\right\} + c_3 - 1 \\ &\leq \sum_{j=1}^N a_{\varphi, j} \beta^{(j)}(\alpha n) + c_3 - 1, \end{aligned}$$

where the coefficients $a_{\varphi, j}$ depend only on the composition operator φ . We can therefore conclude that there exist constants $a_1^{(i)}, \dots, a_N^{(i)}$ such that for *any* balanced decomposition, regardless of the composition operator,

$$b(Z_G^{(i)}) \leq \sum_{j=1}^N a_j^{(i)} \beta^{(j)}(\alpha n) + c_3 - 1.$$

Maximizing over all n -vertex graphs G , we have

$$\beta^{(i)}(n) \leq \sum_{j=1}^N a_j^{(i)} \beta^{(j)}(\alpha n) + c_3 - 1, \quad (11)$$

for $i = 1, \dots, N$. This system of inequalities can be expressed as

$$\bar{\beta}(n) \leq A\bar{\beta}(\alpha n) + \bar{c}, \quad (12)$$

where $\bar{\beta}(n) = (\beta^{(1)}(n), \dots, \beta^{(N)}(n))$, $A = (a_j^{(i)})$ is an $N \times N$ matrix and \bar{c} is a constant N -vector. By standard arguments, it is easy to verify that each $\beta^{(j)}$ is polynomially-bounded. By (9), $\beta(n)$ and, hence, $b(Z_G)$ are polynomially bounded.

As far as constructing Z_G , we can use the given $O(n)$ -size parse tree (T, δ) of G and equations (6), (7), and (8) to calculate Z_G in a bottom-up fashion. A leaf can be processed in $O(1)$ time using exhaustive enumeration. An internal node can be processed as soon as its children have been processed. This can be done in polynomial time if the functions involved have polynomially many breakpoints, as explained earlier. Since there are $O(n)$ nodes in the tree, the computation takes polynomial time. \square

5 Parametric Search Problems

We are interested in search problems where we must find the λ -value, called the *critical point*, at which a particular event occurs in Z_G^P . For the remainder of this section, we restrict our attention to problems where weights are *linear* functions of λ and Z_G^P is a lower envelope (i.e., “opt” is “min” in equation (2)). The following three problems are instances of parametric search:

- (P1) Given a value λ_1 and a subgraph H which is optimum at λ_1 , find the largest $\lambda^* \geq \lambda_1$ such that $Z_G^P(\lambda) = \text{Val}_G(H, \lambda)$ for all $\lambda \in [\lambda_1, \lambda^*]$.
- (P2) Given $t \in \mathbf{R}$, find $\lambda^* \in \mathbf{R}$ such that $Z_G^P(\lambda^*) = t$. We assume that such a λ^* exists.
- (P3) Find λ^* such that $Z_G^P(\lambda^*) = \max_{\lambda} Z_G(\lambda)$.

Each of the above problems has important applications. Problem (P1) is the *sensitivity analysis* problem [Gus83], problem (P2) arises in *minimum-ratio optimization* [Meg79], and problem (P3) arises in *Lagrangian relaxation* [Fis81]. We shall briefly discuss the last of these three problems. Lagrangian relaxation is a heuristic approach to problems with difficult constraints. For instance, consider our original optimization problem (equation (1)), where in addition to weight functions w_V and w_E , every $v \in V(G)$ ($e \in E(G)$) has a *size* $s_V(v)$ ($s_E(e)$). The problem is to solve (1) subject to the knapsack-like constraint

$$\sum_{v \in V(H)} s_V(v) + \sum_{e \in E(H)} s_E(e) \leq t$$

where $t \in \mathbf{R}$. Even if the unconstrained problem is polynomially-solvable, the constrained one may be NP-hard. Such is the case, for example, for the dominating set problem on trees [McPe90]. The problem can be relaxed by incorporating the complicating constraint into the objective function using a Lagrange multiplier λ . The result is a parametric problem of the form (2) where all weights are linear functions of λ [Fis81]. It is well known that, for all $\lambda \geq 0$, $Z_G(\lambda)$ is a lower bound on the value of the optimum solution to the constrained problem. The greatest lower bound is obtained by solving a problem of the form (P3). Such a lower bound can be used with great effectiveness in branch-and-bound schemes [Fis81].

The main result of this section is the following theorem.

Theorem 5.1 *Let $w \geq 1$ and let P be a predicate that is regular with respect to some w -adequate set of composition operators \mathcal{R} . Then, given any weighted n -vertex graph $G \in \Gamma_w$, together with a balanced parse tree $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$ of G , problems (P1), (P2), and (P3) can be solved in $O(n \log n)$ time.*

We shall make use of the following lemma.

Lemma 5.2 *Suppose that $\mathcal{F} = \mathcal{R}^*$ is a class of decomposable graphs and that P is property that is regular with respect to \mathcal{R} . Then, for each of (P1), (P2), and (P3), there exists an oracle which, given a linear-size parse tree of $G \in \mathcal{F}$, answers the following question in linear time: Given $\lambda_0 \in \mathbf{R}$, determine whether or not the critical point λ^* of Z_G^P satisfies $\lambda^* \geq \lambda_0$.*

Proof. All three oracles use Theorem 3.4 to compute a subgraph H_0 of G that is optimum at λ_0 . The subsequent steps depend on the problem.

For (P1), the oracle returns “yes” if $\lambda_0 < \lambda_1$. Otherwise, the oracle uses Theorem 3.4 to compute a subgraph H_1 of G that is optimum at λ_1 . If $\text{Val}_G(H_1, \lambda_0) > \text{Val}_G(H_0, \lambda_0)$, it returns “no”; otherwise, it returns “yes”. For (P2), the oracle examines the slope of the line $\text{Val}_G(H_0, \lambda)$. Suppose it is positive. Then, if $\text{Val}_G(H_0, \lambda_0) > t$, it returns “no”; otherwise, it returns “yes”. The case where the slope of $\text{Val}_G(H_0, \lambda)$ is nonpositive is handled analogously. For (P3), the oracle examines the slope of the line $\text{Val}_G(H_0, \lambda)$. If it is negative, it returns “no”; otherwise, it returns “yes”. \square

We will also need one further result, due to Cole [Cole87]. We assume some familiarity with *combinational circuits* as discussed, say, in [CLR90]. A combinational circuit \mathcal{B} is a directed acyclic graph whose nodes are *combinational elements*, and where an edge from element e_1 to element e_2 implies that the output of e_1 is an input to e_2 . Combinational elements are computational units that have a constant number of inputs and outputs and that perform well-defined operations. Typically, such operations are simple (e.g., adding or finding the minimum of two numbers); however, for conceptual purposes, we shall find it convenient to also deal with circuits whose elements perform more complex tasks, such as taking the lower envelope of two functions. We refer to $|V(\mathcal{B})|$ as the *size* of \mathcal{B} . Elements of zero fan-in are *inputs*; elements of zero fan-out are *outputs*. An element is said to be *active* if all its inputs are known, but the associated operation has not been carried out yet. We shall say that an element has been *resolved* if the associated operation has been carried out. Suppose we have a weight function $\omega : V(\mathcal{B}) \rightarrow \mathbf{R}$. The *active weight*, W , of \mathcal{B} is the sum of the weights of its active elements. An *oracle with respect to ω* is a procedure that is guaranteed to resolve a set of active elements whose total weight is at least $W/2$.

Lemma 5.3 [Cole87] *Let \mathcal{B} be a combinational circuit of size M and depth $f(M)$. Let $d_{\min} = \min\{d_I, d_O\}$, where d_I (d_O) denotes the maximum fan-in (fan-out) of an element of \mathcal{B} . Then, there exists a weight function ω such that \mathcal{B} can be evaluated with $O(f(M) \log d_{\min} + \log M)$ calls to an oracle with respect to ω .*

Proof of Theorem 5.1. We shall use ideas from [Meg83, Cole87]. As in section 4, let \mathcal{A} denote the algorithm for computing z_G described in section 3.2, and let Z_G and $Z_G^{(i)}$ be the parametric analogs of z_G and $z_G^{(i)}$ (see equations (6) and (7)). The proof proceeds by first showing how to hard-wire \mathcal{A} into a combinational circuit \mathcal{B}_0 whose inputs are the vertex and edge weights at λ and whose output is $Z_G(\lambda)$. Next, we construct a parametric version of \mathcal{B}_0 , which we refer to as \mathcal{B} , whose inputs are the vertex and edge weight functions, and whose output is the function Z_G . Finally, we devise a search algorithm \mathcal{A}_S that guides the execution of \mathcal{B} to determine the behavior of Z_G in the neighborhood of the critical point λ^* .

The elements of \mathcal{B}_0 are adders, subtractors, and “min” gates (i.e., gates whose output is the minimum of their two input numbers). We obtain \mathcal{B}_0 from a balanced parse tree (T, δ) of G and from the multiplication tables for the operators in \mathcal{R} and the equivalence classes C_1, \dots, C_N as follows. For $v \in V(T)$, let G_v denote $\gamma_T(v)$. Note that for every $v \in V(T)$ and every i , $1 \leq i \leq N$, there exists a $O(1)$ -size circuit

$\text{CT}(\varphi, i)$, where $\varphi = \delta(v)$, that computes $z_{G_v}^{(i)}$, given the appropriate inputs. These inputs will be edge and vertex weights of φ if $\varphi \in \mathcal{R}_0$ (in which case the circuit is computing the weight of an optimum subgraph by exhaustive enumeration); or they will be the $z_{G_j}^{(i)}$'s for graphs corresponding to children of v (in which case, the circuit evaluates equation (5) from section 3.2). In either case, the structure of $\text{CT}(\varphi, i)$ depends only on $\text{MT}(\varphi, i)$. There is also a $O(1)$ -size circuit that computes the value of z_G from the values of the $z_{G_u}^{(i)}$'s, where u is the root of T . All of these circuits can be constructed entirely out of min gates, adders, and subtractors of fan-in at most two. By following the structure of T , these various $O(1)$ -size circuits can be assembled to obtain a circuit \mathcal{B}_0 of size $O(n)$ and depth $O(\log n)$. We leave the details to the reader.

To construct the parametric circuit \mathcal{B} , we modify circuit \mathcal{B}_0 by replacing its elements, which manipulate real numbers, with elements that manipulate functions of λ and by replacing its inputs with the corresponding functions of λ . Thus, the min gates will compute lower envelopes of their inputs and the adders will construct the sum of their input functions (this is similar to the construction used in the proof of Theorem 4.2). The size and depth of \mathcal{B} are identical to those of \mathcal{B}_0 . For $v \in V(\mathcal{B})$, we write $S_v(\lambda)$ to denote the output of v at λ . We leave it to the reader to verify that the inputs and outputs of every circuit element are piecewise linear concave functions, and that $Z_G = S_u$, where u is the output node of \mathcal{B} .

The search algorithm \mathcal{A}_S simulates the execution of \mathcal{B} to find the structure of Z_G within a certain closed interval I^* with the properties that (1) $\lambda^* \in I^*$ and (2) Z_G is a line in I^* . At all times, \mathcal{A}_S maintains an interval $I = [\lambda_L, \lambda_R]$ such that $\lambda^* \in I$; initially $\lambda_L = -\infty$ and $\lambda_R = +\infty$. We shall say that \mathcal{A}_S has *resolved* an element v of \mathcal{B} if S_v is a line in the interval I and the equation of this line has been computed. An unresolved element v is *active* if v is an input, or all elements u such that $(u, v) \in E(\mathcal{B})$ have been resolved. At any stage of the simulation of \mathcal{B} algorithm \mathcal{A}_S chooses which active elements to resolve according to a certain weight function ω defined on $V(\mathcal{B})$ and a procedure **RESOLVE**, which is described below.

Procedure **RESOLVE**(λ_L, λ_R)

Step 1. Let A be the set of active nodes of \mathcal{B} . Let A_1 be the set of adders and subtractors in A and let A_2 be the set of min gates in A .

Step 2. Resolve each $v \in A_1$ by constructing S_v in its entirety in the interval $I = [\lambda_L, \lambda_R]$.

Step 3. For each $v \in A_2$, construct S_v in its entirety in the interval I , by taking the lower envelope of the inputs S_{u_1} and S_{u_2} . Let λ_v denote the single breakpoint of this function in the interval (λ_L, λ_R) ; if S_v has no breakpoints in this open interval, set $\lambda_v = +\infty$.

Step 4. Let $U = \{v \in A_2 : \lambda_v \in (\lambda_L, \lambda_R)\}$. Resolve each $v \in A_2 - U$.

Step 5. If U is empty, stop. Otherwise, compute the weighted median λ_{v_m} of the set $\{\lambda_v : v \in U\}$, where the weight of λ_v is the weight $\omega(v)$ of the corresponding element of \mathcal{B} . Use the oracle of Lemma 5.2 to determine whether or not $\lambda^* \geq$

λ_{v_m} . If the answer to this call is “yes”, then set $\lambda_L = \lambda_{v_m}$ and resolve all elements $v \in U$ such that $\lambda_v \leq \lambda_{v_m}$. Otherwise, set $\lambda_R = \lambda_{v_m}$ and resolve all elements $v \in U$ such that $\lambda_v \geq \lambda_{v_m}$.

\mathcal{A}_S calls procedure RESOLVE repeatedly until all the elements of \mathcal{B} have been resolved. We now argue that RESOLVE is a $O(n)$ -time oracle with respect to any weight function ω . Observe that since $|V(\mathcal{B})| = O(n)$, the sets A , A_1 , A_2 , and U of Steps 1–5 have cardinality $O(n)$. Step 1 takes $O(n)$ time. Constructing each S_v function in Steps 2 or 3 takes $O(1)$ time, since the inputs to the corresponding element have no breakpoints inside I . Thus, Steps 2 and 3 take $O(n)$ time. Since the S_v ’s computed in Step 2 are sums of functions that are linear within I , the S_v ’s themselves are linear within I . Thus, all the corresponding v ’s are resolved. Each S_v constructed in Step 3 has at most one breakpoint $\lambda_v \in (\lambda_L, \lambda_R)$. Step 4 finds, in $O(n)$ time, the set U of gates that have breakpoints in (λ_L, λ_R) . All other min gates are, by definition, resolved. If U is empty, then all the min gates in A are resolved. If U is nonempty, Step 5 finds a weighted median, in $O(n)$ time [CLR90], and does one oracle call, which, by Lemma 5.2, takes $O(n)$ time. If the answer is “yes”, then for every $v \in U$ such that $\lambda_v \leq \lambda_{v_m}$, S_v has no breakpoints in $(\lambda_{v_m}, \lambda_R)$; if the answer is “no”, then for every $v \in U$ such that $\lambda_v \geq \lambda_{v_m}$, S_v has no breakpoints in $(\lambda_L, \lambda_{v_m})$. In either case, Step 4 guarantees that at least a weighted half of the min gates in U will be resolved and it preserves the invariant that $\lambda^* \in I$ (see [Cole87] for a similar argument). Since all the gates in $A - U$ are resolved, we conclude that RESOLVE is a $O(n)$ -time oracle with respect to ω .

The correctness of RESOLVE implies that at the end of the simulation, I will be an interval containing λ^* within which Z_G is a line whose equation is known. At this point, computing λ^* will be straightforward. Since the fan-in of any element is at most 2 and RESOLVE takes $O(n)$ time, Lemma 5.3 implies that \mathcal{A}_S will resolve all the elements of \mathcal{B} in $O(n \log n)$ time. \square

6 Discussion

The main results of this paper, Theorems 4.2 and 5.1, rely on the existence of balanced decompositions and parse trees. The key to constructing these is procedure DECOMPOSE, presented in the proof of Theorem 3.2, whose main component is an algorithm delivering a partition satisfying Theorem 2.1. A linear-time separator algorithm is easily obtainable from what is known as the *embedding w -tree* of the graph [ArPr89] (see, e.g., [FeMe90]). Since such embeddings can be constructed in linear time for graphs of tree-width 1, 2, or 3 [MaTh91], we can construct partitions of such graphs in $O(n)$ time, and balanced parse trees in $O(n \log n)$ time. Things are more complicated for graphs of tree-width $w > 3$. It is easy to see that all of our results are valid if instead of a partition satisfying Theorem 2.1, we have a partition (B_1, B_2, B_3, S) satisfying all the conditions of this theorem, except that we only guarantee that $|B_i \cap Q| \leq (1 - d_0(w))|Q - S|$, for some function $d_0(w)$ such that $0 < d_0(w) \leq 1/2$ for $w > 3$. We refer to the separator S in this partition as an *approximate separator*. Lagergren [Lag91] has shown how an approximate separator

can be produced in $O(n \log n)$ time (see also [Reed92]). The approximate separator algorithm can be used to construct parse trees of size $O(n)$ and height $O(\log n)$ in $O(n \log^2 n)$ time.

Theorem 4.2 implies that if vertex and edge weights are bounded-degree polynomials in λ , then problems (P1)–(P3) can be solved in polynomial time by simply constructing Z_G . Unfortunately, we do not see any easy way to extend the results of section 5 to problems where costs are polynomial functions of λ . There are, however, two search problems for which we have $O(n)$ -time algorithms.

(P4) Given $\lambda_0 \in \mathbf{R}$, find a $\lambda^* > \lambda_0$ such that Z_G^P has no breakpoints in (λ_0, λ^*) .

(P5) Find λ_∞ such that Z_G^P has no breakpoints in $(\lambda_\infty, +\infty)$.

Problem (P4) is a simplified version of the sensitivity analysis problem, while problem (P5) is the *steady state problem* [Ata85, FeSl89]. Observe that, given a solution to (P5), we can, in $O(n \log n)$ time, find the last breakpoint of Z_G for the case where weights are linear. This simply involves computing λ_∞ and then using a slight modification of the algorithm for problem (P1) (see [FeSl89]). Very similar techniques can be used to find the first breakpoint.

In the next theorem we assume, as in section 4, a model of computation where computing the roots of a d -th degree polynomial is a primitive operation.

Theorem 6.1 *Let $w \geq 1$ and let P be a predicate that is regular with respect to some w -adequate set of composition operators \mathcal{R} . Then, given any weighted n -vertex graph $G \in \Gamma_w$, together with a $O(n)$ -size parse tree $(T, \delta) \in \mathcal{T}_{\mathcal{R}}$ of G , problems (P4) and (P5) can be solved in $O(n)$ time,*

Proof. As in the proof of Theorem 5.1, let \mathcal{B} be the circuit associated with (T, δ) . Since (T, δ) is of size $O(n)$, so is \mathcal{B} . The algorithms for solving problems (P4) and (P5) are similar. Both involve simulating \mathcal{B} , resolving its elements in a bottom-up fashion. We will only describe the algorithm for (P5), the steady-state problem, in some detail and leave the solution to (P4) as an exercise. We shall refer to the algorithm for (P5) as \mathcal{A}_∞ .

The goal of \mathcal{A}_∞ is to discover the behavior of \mathcal{B} at infinity. At all times, \mathcal{A}_∞ maintains an interval $I = (\lambda_L, \infty)$; initially, $\lambda_L = -\infty$. We say that an element v of \mathcal{B} is resolved if S_v is a polynomial function in I and the equation of this polynomial is known. An element is said to be active if it is not resolved and it is either an input element or both of its inputs are resolved. Algorithm \mathcal{A}_∞ resolves active elements of \mathcal{B} one by one, in any order, until all elements are resolved. An active element v can be resolved in $O(1)$ time as follows. Suppose that the inputs to v are u_1 and u_2 . If v is an adder or a subtractor, construct S_v within I directly from S_{u_1} and S_{u_2} . This function clearly has no breakpoints within I . If v is a min gate, construct S_v within I by taking the lower envelope of S_{u_1} and S_{u_2} . Let $\lambda_{v_1}, \dots, \lambda_{v_k}$, $k \leq d$, be the breakpoints of S_v within I . Set λ_L equal to $\max\{\lambda_L, \lambda_{v_1}, \dots, \lambda_{v_k}\}$. After this is done, S_v will have no breakpoints in (λ_L, ∞) . Once the output element v of \mathcal{B} is resolved, we return $\lambda_\infty = \lambda_L$. \square

The restriction to graphs of bounded tree-width seems to be important in achieving our bounds. Without it, some problems do indeed have an exponential number of breakpoints in the worst case [Car83]. However, the bound of Theorem 4.2 can probably be sharpened considerably in certain special cases. A natural candidate for further study is the maximum independent set problem. Improving the running times of the algorithms for the search problems described in Section 5 is another intriguing problem. We see no obvious reason why $\Omega(n \log n)$ should be a lower bound for the solution of these problems

Acknowledgements

The first author thanks Bruno Courcelle for his hospitality in Bordeaux, and Hans Bodlaender, Jens Lagergren, and Andrzej Proskurowski for valuable discussions during the early stages of this work.

References

- [AbFe92] K. Abrahamson and M.R. Fellows. Finite automata, bounded treewidth, and well-quasiordering. *Preprint* (1992).
- [ArPr89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k -trees. *Discr. Appl. Math.*, 23:11–24 (1989).
- [ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340.
- [Ata85] M. Atallah. Dynamic computational geometry. *Comp. & Maths. with Appls.*, 11(12):1171–1181, 1985.
- [BLW87] M.W. Bern, E.L. Lawler, and A.L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.
- [BPT88] R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms from predicate-calculus descriptions of problems on recursively-constructed graph families. Manuscript, 1988. To appear in *Algorithmica*.
- [Bod87] H.L. Bodlaender. Dynamic programming on graphs with bounded tree-width. Technical Report RUU-CS-88-4, University of Utrecht, 1988. Extended Abstract in *Proceedings of ICALP88*.
- [Bod88] H.L. Bodlaender. Some classes of graphs with bounded tree-width. *Bulletin of the EATCS*, **36** (1988), 116-126.
- [Car83] P. Carstensen. Complexity of some parametric integer and network programming problems. *Math. Programming*, 26:64–75, 1983.
- [Cole87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. Assoc. Comput. Mach.*, 34(1):200–208, 1987.

- [Cou90] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
- [CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Mass., 1990.
- [EaRo89] B.C. Eaves and U.G. Rothblum. A theory on extending algorithms for parametric problems. *Mathematics of Operations Research*, 14(3):502–533, 1989.
- [FeSl89] D. Fernández-Baca and G. Slutzki. Solving parametric problems on trees. *J. Algorithms*, 10:381–402 (1989).
- [FeMe90] D. Fernández-Baca and A. Medepalli. Parametric module allocation on partial k -trees. Technical Report 90-25, Department of Computer Science, Iowa State University, December 1990.
- [Fis81] M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18 (1981).
- [GGT89] G. Gallo, M.D. Grigoriades, and R.E. Tarjan. A fast parametric maximum flow algorithm and its applications. *SIAM J. Computing*, 18(1):30–55, 1989.
- [Gus80] D. Gusfield. *Sensitivity analysis for combinatorial optimization*. Technical Report UCB/ERL M80/22, University of California, Berkeley, May 1980.
- [Gus83] D. Gusfield. Parametric combinatorial computing and a problem in program module allocation. *J. Assoc. Comput. Mach.*, 30(3):551–563, July 1983.
- [vHKRW89] C.P.M. van Hoesel, A.W.J. Kolen, A.H.G. Rinooy and A.P.M. Wagelmans, *Sensitivity analysis in combinatorial optimization: a bibliography*. Report 8944/A, Econometric Institute, Erasmus University Rotterdam, 1989.
- [GaJo79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [Lag90] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of 31st Annual Symposium on Foundations of Computer Science*, pp. 173–182 (1990).
- [Lag91] J. Lagergren. Algorithms and minimal forbidden minors for tree-decomposable graphs. (Doctoral Dissertation) Technical Report TRITA-NA-9104, Royal Institute of Technology, Sweden, March 1991.
- [MaTh91] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22 (1991).
- [McPe90] J. McHugh and Y. Perl. Best location of service centers in a treelike network under budget constraints. *Discrete Mathematics*, 86:199–214 (1990).
- [Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424 (1979).

- [Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.*, 30(4):852–865, 1983.
- [Mur80] K. Murty. Computational complexity of parametric linear programming. *Math. Programming*, 19:213–219, 1980.
- [Reed92] B.A. Reed. Finding approximate separators and computing tree width quickly. To appear in *STOC 92*.
- [RoSe86] N. Robertson and P.D. Seymour. Graph minors II: Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.
- [RoSe] N. Robertson and P.D. Seymour. Graph minors XIII: The disjoint paths problem. *To appear*.
- [vLe90] J. van Leeuwen. Graph Algorithms. In J. van Leeuwen (ed.) *Handbook of Theoretical Computer Science*, MIT Press, Cambridge, Mass., 1990.
- [Sha87] M. Sharir. Almost linear upper bounds on the length of Davenport-Schinzel sequences. *Combinatorica*, 7(1):131–143 (1987).
- [Wim87] T.V. Wimer. Linear algorithms on k -terminal graphs. Ph.D. Thesis, Report No. URI-030, Clemson University (1987).
- [ZhGo91] B. Zhu and W. Goddard. An algorithm for outerplanar graphs with parameter. *J. Algorithms*, 12:6657–6662 (1991).



IOWA STATE UNIVERSITY

OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

SCIENCE
with
PRACTICE

Tech Report: TR 92-08
Submission Date: April 22, 1992