Practical algorithms on partial k-trees with an application to domination-like problems

Jan Arne Telle & Andrzej Proskurowski

CIS-TR-93-04
February 1993

Department of Computer and Information Science
University of Oregon

Practical algorithms on partial k-trees with an application to domination-like problems

Jan Arne Telle, Andrzej Proskurowski
Department of Computer and Information Science
University of Oregon, Eugene, Oregon 97403, USA

Abstract

Many NP-hard problems on graphs have polynomial, in fact usually linear, dynamic programming algorithms when restricted to partial k-trees (graphs of treewidth bounded by k), for fixed values of k. We investigate the practicality of such algorithms, both in terms of their complexity and their derivation, and account for the dependency on the treewidth k. We define a general procedure to derive the details of table updates in the dynamic programming solution algorithms. This procedure is based on a binary parse tree of the input graph. We give a formal description of vertex subset optimization problems in a class that includes several variants of domination, independence, efficiency and packing. We give algorithms for any problem in this class, which take a graph G, integer k and a width k tree-decomposition of G as input, and solve the problem on G in $O(n2^{4k})$ steps.

1 Introduction

A graph G is a k-tree if it is a complete graph on k vertices or if it has a vertex $v \in V(G)$ whose neighbors induce a clique of size k and $G - \{v\}$ is again a k-tree. The class of partial k-trees (the subgraphs of k-trees) is identical to the class of graphs of treewidth bounded by k. Many natural classes of graphs have bounded treewidth. These classes are of algorithmic interest because many optimization problems, while inherently difficult (NP-hard) for general graphs are solvable in linear time on partial k-trees, for fixed k. These solution algorithms have two main steps, first finding a parse tree (tree-decomposition of width k) of the input graph, and then computing the solution by a bottom-up traversal of the parse tree. For the first step, Bodlaender [8] has given a linear algorithm deciding if a graph is a partial k-tree and if so finding a tree-decomposition of width k, for fixed k. Unfortunately, the complexity of this

algorithm as a function of the treewidth does not make it practical for larger values of k. For $k \leq 4$, however, practical algorithms based on graph rewriting do exist for the first step [4, 16, 19]. In this paper we investigate the complexity of the second step when k is not fixed. There are many approaches to finding a template for the design of algorithms on partial k-trees with time complexity polynomial, or even linear, in the number of vertices [17, 2]. As a rule, these approaches have tried to encompass as wide a class of problems as possible, often at the expense of increased complexity in k and also at the expense of simplicity of the resulting algorithms. Results giving explicit practical algorithms in this setting are usually confined to a few selected problems on either partial 1-trees or partial 2-trees [20, 14, 22]. In this paper we try to cover the middle ground between these extremes.
In the next section, we present a binary parse tree of the input graph which can easily be derived from a tree-decomposition. This parse tree is based on very simple graph operations which, as we show, will ease the derivation of practical solution algorithms for many problems. In section 3, we give a formal description of vertex subset optimization problems in a class that includes several variants of domination, independence, efficiency and packing. Then, in section 4, we present an algorithm template for any problem in this class. These algorithms take a graph G, integer k and a width k tree-decomposition of G as input, and solve the problem on G in $\mathcal{O}(n2^{4k})$ steps. Since these problems are NP-hard in general and a graph on n vertices is an n-tree, we should not expect polynomial dependence on k. In the last section we discuss and give some extensions of the algorithms.

2 Practical algorithms on partial k-trees

We use standard graph terminology [9]. For a vertex $v \in V(G)$ of a graph G, let $N_G(v) = \{u : uv \in E(G)\}$, and $N_G[v] = N_G(v) \cup \{v\}$, be its open and closed neighborhoods, respectively. Let $G[S]$ denote the graph induced in G by a subset of vertices $S \subseteq V(G)$. A partial k-tree G is a partial graph of a k-tree H, meaning $V(G) = V(H)$ and $E(G) \subseteq E(H)$. A k-tree H has a perfect elimination ordering of its n vertices $peo = v_1, ..., v_n$ such that $\forall i : 1 \leq i \leq n-k$ we have $B_i = N_H[v_i] \cap \{v_i, ..., v_n\}$ inducing a $(k+1)$-clique in H. We call $B_i, 1 \leq i \leq n-k$ the $(k+1)$-*bag* of v_i in G under *peo* and each of its k-subsets is similarly called a k-bag of G under *peo*. The remaining definitions in this section are all for given G, H and $peo = v_1, ..., v_n$ as above.

We define a *peo-tree* P of G under *peo* with the node set $\{B_1, ..., B_{n-k}\}$. The node B_i has as its parent in P the node B_j such that $j > i$ is the minimum one with $|B_i \cap B_j| = k$. The $(k+1)$-ary peo-tree P is a clique tree of H and also a width k tree-decomposition of both G and H (see Figure 1 for an example.)

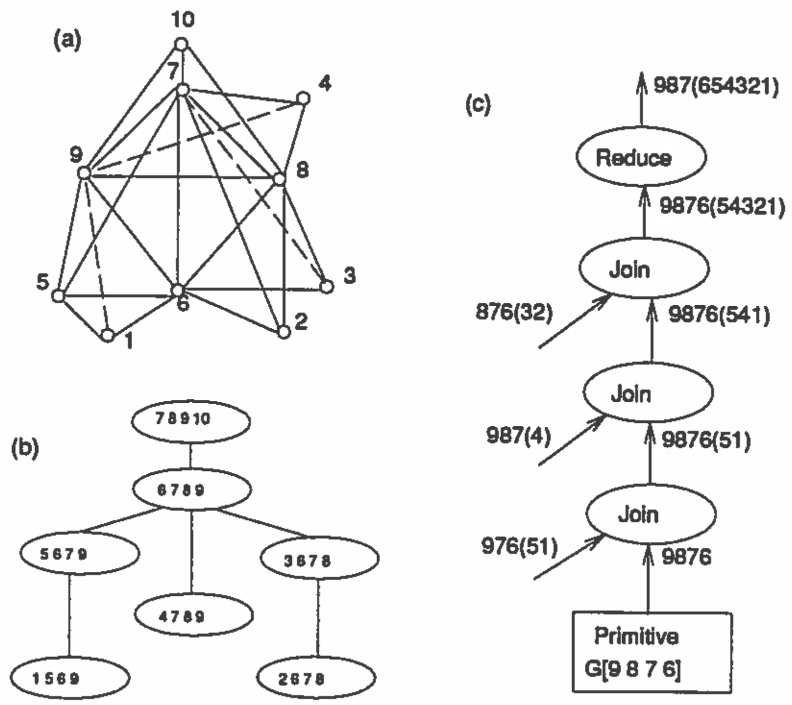

Figure 1: (a) A partial 3-tree G, embedded in a 3-tree H, dashed edges in $E(H) - E(G)$, with peo=1,2,3,4,5,6,7,8,9,10. (b) Its peo-tree P. (c) Part of its binary parse tree T defined by node $B_6 = \{6, 7, 8, 9\}$ of P. Child-to-parent arcs of T labeled by $V(G_{child})$ with non-sources in parenthesis.

We sketch an algebra on simple graphs needed to define a binary parse tree T of G based on the peo-tree P. Let a graph with k distinguished vertices (also called sources, terminals, boundaries) have type G_k. For our purposes the set of sources will always be $(k+1)$-bags or k-bags of G. Primitive graphs are $G[B]$ for some $(k+1)$-bag B and have type G_{k+1}. The two operations are Reduce: $G_{(k+1)} \rightarrow G_k$ and Join: $G_{k+1} \times G_k \rightarrow G_{k+1}$. Reduce takes the vertex to be eliminated and discards it as a source. Join takes the union of its two argument graphs, where the sources of the second graph (a k-bag) are a subset of the sources of the first graph (a $(k+1)$-bag); these are the only shared vertices, and the two graphs agree on edges between sources.

The construction of the binary parse tree T is based on the peo-tree P. A node B_i of P, with c_i children and parent $p(B_i)$ defines a leaf-towards-root path of T. The path starts with the primitive graph $G[B]$ as the leaf endpoint, has c_i Join operations as interior nodes and terminates with a node of a Reduce operation. The Reduce operation discards the vertex v_i as a source, and its node is the second child of the node of a Join operation defined by $p(B_i)$ (see Figure 1 for an example.)

Note that T is indeed a parse tree of G since the primitive graphs in T contain all vertices and edges of G, while Join and Reduce merely identify vertices of the primitive graphs, in the order given by P, to form G. The root of T is thus a Reduce node representing G with sources $\{v_{n-k+1}, ..., v_n\}$. Since P is a tree with $n-k$ nodes, the Binary Parse Tree T of G has

- $n-k$ Primitive leaves, one for each $(k+1)$-bag
- $n-k$ Reduce operations, one for each vertex eliminated
- $n-k-1$ Join operations, one for each edge of T

A dynamic programming solution algorithm for a problem on G will follow a bottom-up traversal of T. As usual, at each node u of T a data structure *table* is kept of optimal solutions restricted to G_u, the subgraph of G represented by the subtree of T rooted at u. The table of a leaf is filled as a base case, the table of an interior node is filled based on tables of its children and the overall solution is obtained from the table at the root. The following information will complete the algorithm description for a given problem

- Description of Tables
- Operations Initialize-Primitive-Table
- Operation Join-Tables
- Operation Reduce-Table
- Operation Root-Optimization

The dynamic programming strategy on partial k-trees of [5], differed from the above primarily in that a vertex v_i with $(k+1)$-bag B was eliminated by combining tables of all $k+1$ k-bags in B in a single $(k+1)$-ary operation. Assuming the table for a k-bag has index set I_k, this operation has complexity $\Omega(|I_k|^{k+1})$ when all combinations of entries from each table are considered. Intuitively, the binary parse tree approach described above replaces a single such $(k+1)$-ary operation by at most k pairs of binary Join and Reduce operations, for complexity $\mathcal{O}(k|I_k||I_{k+1}|)$. Next we show how the description and the derivation of the correct procedure for updating a table based on the tables of children is simplified by using the binary parse tree T.

Many strategies have been proposed for solving problems on graphs of bounded treewidth using a variant of the dynamic programming described above. For any such strategy, an algorithm for a given problem must describe the tables involved and

also describe how tables are updated. We can classify these strategies by whether there is a procedure for automatic (mechanical) construction of a solution algorithm from a formal description of the problem, or such an algorithm has to be constructed "by hand".
The EMSOL approach [3] is an automatic technique. A linear time algorithm solving a given problem can be constructed automatically from the problem description in a certain logic formalism. Although very strong for showing assymptotic complexity results, this technique is unsatisfactory for practical algorithm design since the resulting complexity in k involves towers of powers of k. Currently, there is no automatic algorithm design strategy giving algorithms which are practical for increasing values of k without hand derivation of a large part of the algorithm [6, 8, 12, 23, 1, 10].

Hand derivation of algorithms involves definition of table indices. Each table index represents an equivalence class of solutions to subproblems, equivalent in terms of forming parts of larger solutions. A solution to a subproblem at a node u of the parse tree T is restricted to G_u. This subproblem solution interacts with the solutions to larger problems only through the sources of G_u, a separator of G. Equivalent solutions affect the separator in the same manner, and hence we can define a *separator state* for each equivalence class (table index). A candidate set of separator states is verified by the correctness proof of table update procedures for all operations involved. The introduction of the operations Reduce and Join greatly simplifies this verification process.

For many problems, we can define a set L of vertex states, that represent the different ways that a solution to a subproblem can affect a source vertex, such that $|L|$ is independent of both n and k. A separator with k vertices has a separator state for each k-vector of vertex states so its table index set has size $|I_k| = |L|^k$. In the next section we first present a class of such problems and then derive $O(n|L|^{2k})$ dynamic programming solution algorithms for partial k-trees.

3 Formalization of domination-like problems

In this section, we formalize a class of problems to which we find realistic linear solution algorithms on partial k-trees given with a tree-decomposition of width k. Given a set $S \subseteq V(G)$ of *selected* vertices, we assign a *state* to each vertex of G as follows:

$$state_S(v) = \begin{cases} \rho_i & \text{if } v \in V(G) - S \text{ and } |N_G(v) \cap S| = i \\ \sigma_i & \text{if } v \in S \text{ and } |N_G(v) \cap S| = i \end{cases}$$

We also use the abbreviations:

Term	Expressed in our formalism
Dominating	ρ_0 not a final state
Independent	σ_0 only final σ-state
Perfect Dominating	ρ_1 only final ρ-state
Nearly Perfect	ρ_0 and ρ_1 only final ρ-states
Efficiency of $[F]$-set	$\max_{S \text{ is } [F]\text{-set}} \lvert\{v : states_S(v) = \rho_1\}\rvert$
Total P	effect of P on ρ-states is extended to σ-states

Table 1: The established terminology and our formalism

$$\rho_{\geq i} \overset{df}{=} \{\rho_i, \rho_{i+1}, \ldots\}$$

$$\sigma_{\geq i} \overset{df}{=} \{\sigma_i, \sigma_{i+1}, \ldots\}$$

Mnemonically, σ and ρ represents a vertex selected for S or rejected from S, respectively, with the subscript indicating the number of neighbors it has in S. A variety of graph parameters can be defined by optimizing over subsets S while allowing only a specific set F as *final* states of vertices. For example, S is a dominating set if state ρ_0 is not allowed for any vertex, giving the final states $F = \{\rho_{\geq 1}, \sigma_{\geq 0}\}$. The set $L = \{\rho_0, \rho_{\geq 1}, \sigma_{\geq 0}\}$ of *legal* states with respect to F is a superset of the final states containing also, for ρ-states and σ-states separately, any state with an integer subscript smaller than that of a final state. These states are needed for algorithmic purposes since a vertex may acquire additional neighbors from S during the bottom-up traversal of the parse tree. Optimization problems over these sets maximize or minimize the size of the set of vertices with given states M and are denoted $minM[F]$ or $maxM[F]$. For instance, $min\{\sigma_{\geq 0}\}[\rho_{\geq 1}, \sigma_{\geq 0}]$ is the minimum size dominating set problem.

Definition Given a set F of final states as above, any $S \subseteq V(G)$ such that $\forall v \in V(G) : states_S(v) \in F$ is an **[F]-set.** For $M \subseteq F$, the problem of minimizing (or maximizing) $|\{v : states_S(v) \in M\}|$ over all $[F]$-sets S is denoted by **minM[F]** (or $maxM[F]$), and the corresponding parameter for a graph G by **minM[F](G)** (or $maxM[F](G)$). The set of **legal states** with respect to F is $L = F \cup \{\rho_i \notin F : \rho_j \in F \wedge j > i\} \cup \{\sigma_i \notin F : \sigma_j \in F \wedge j > i\}$. The sets M, F and L should all contain as few explicit states as possible (i.e. they use abbreviated states if necessary) while maintaining the syntactic restriction $M \subseteq F$.

Table 1 relates some of the established terminology to our formalism and Table 2 shows some of the classical vertex subset properties [11] in our formalism.
We also allow vertex weighted versions and directed graph versions of these optimization problems. As an example, we mention the NP-hard problem Total Redundance

Our notation	$\|L\|$	Standard terminology
$[\rho_{\geq 1}, \sigma_{\geq 0}]$-set	3	Dominating Set
$[\rho_{\geq 0}, \sigma_0]$-set	2	Independent Set
$[\rho_0, \rho_1, \sigma_0]$-set	3	Strong Stable Set or 2-Packing
$[\rho_1, \sigma_0]$-set	3	Efficient Dominating Set or Perfect Code
$[\rho_{\geq 1}, \sigma_0]$-set	3	Independent Dominating Set
$[\rho_1, \sigma_{\geq 0}]$-set	3	Perfect Dominating Set
$[\rho_{\geq 1}, \sigma_{\geq 1}]$-set	4	Total Dominating Set
$[\rho_1, \sigma_1]$-set	4	Total Perfect Dominating Set
$[\rho_0, \rho_1, \sigma_{\geq 0}]$-set	3	Near Perfect Set
$[\rho_0, \rho_1, \sigma_0, \sigma_1]$-set	4	Total Near Perfect Set
$[\rho_1, \sigma_0, \sigma_1]$-set	4	Weakly Perfect Dominating Set
$max\{\rho_1\}[\rho_0, \rho_1, \rho_{\geq 2}, \sigma_{\geq 0}]$	4	Efficiency Problem

Table 2: Some classical definitions

[14], where the weight of a vertex v is given by $w(v) = |N(v)| + 1$ and the parameter minimizes the sum of the weights of vertices with state $\sigma_{\geq 0}$ in any $[\rho_{\geq 1}, \sigma_{\geq 0}]$-set. From Table 2 we note that subset property definitions which distinguish vertices by their having zero, one or more selected neighbors attract the most interest. Similarly, the objective functions most studied involve minimizing or maximizing the cardinality of the set of selected vertices, and in fact, for each entry in Table 2 there is an NP-hard problem related to such a parameter. For Independent Dominating Sets it is well-known that both minimizing and maximizing the set of selected vertices are NP-hard problems. The following result shows additional problems for which both the minimization and the maximization of selected vertices are NP-hard.

Theorem 1 *[21] The problems $opt\{\sigma_1\}[\rho_{\geq 1}, \sigma_1]$, $opt\{\sigma_0, \sigma_1\}[\rho_{\geq 1}, \sigma_0, \sigma_1]$, $opt\{\sigma_0\}[\rho_{\geq 2}, \sigma_0]$ and $opt\{\sigma_1\}[\rho_{\geq 2}, \sigma_1]$ with opt replaced by max or min are all NP-hard. For any $p \geq 1, q \geq 0$ the problem of deciding if a graph G has a $[\rho_p, \sigma_q]$-set is NP-complete (for $p = q = 1$ even with G restricted to a planar bipartite graph of maximum degree three) and if the answer is affirmative then $min\{\sigma_q\}[\rho_p, \sigma_q](G) = max\{\sigma_q\}[\rho_p, \sigma_q](G)$.*

4 Algorithm template

In this section we give algorithms for solving any problem in the formalism just described, on a partial k-tree G.

Algorithm-optM[F], where opt is either *max* or *min*.

Input: G, k,tree-decomposition of G of width k

Output: $optM[F](G)$

(1) Find a binary parse tree T of G with Primitive, Reduce and Join nodes.

(2) Initialize Primitive Tables at leaves of T.

(3) Bottom-up traversal of T using Join-Tables and Reduce-Table.

(4) Table-Optimization at root of T gives $optM[F](G)$.

The algorithm follows the binary parse tree T of G as outlined in section 2. For a given graph G and any fixed k, Bodlaender [8] has given an $O(n)$ (exponential in a polynomial in k) algorithm for deciding if the treewidth of G is at most k and in the affirmative case finding a tree-decomposition of G of width k. From this tree-decomposition it is straightforward to find a binary parse tree of G in time $O(nk^2)$, see e.g. [15] for how to find an embedding in a k-tree, then find a *peo* and finally follow the description given in section 2 for constructing the binary parse tree. For a problem given by the final states F, legal states L and optimizing (min or max) the set of vertices with state in M, we next describe the Tables involved and then give details of Table-Initialization, Table-Reduce, Join-Tables and Table-Root-Optimization. Each of the following subsections defines the appropriate procedure, gives the proof of its correctness and analyzes its complexity.

Table Description

Let a node u of the parse tree T represent the subgraph G_u of G with sources $B_u = \{w_1, ..., w_k\}$. The table at node u, $Table_u$, has index set $I_k = \{\mathrm{s} = s_1, ...s_k : s_i \in L\}$, so that $|I_k| = |L|^k$. We define Ψ, with respect to G_u and s, to be the family of sets $S \subseteq V(G_u)$ such that in the graph G_u, for $w_i \in B_u$, $states_S(w_i) = s_i, 1 \leq i \leq k$ and for $v \in V(G_u) - B_u, states_S(v) \in F$. Ψ forms an equivalence class of solutions to the subproblem on G_u, and its elements are called Ψ-sets respecting G_u and s. The value of $Table_u$[s] will be the optimum (max or min) number of vertices in $V(G_u) - B_u$ that have state in M over all Ψ-sets respecting G_u and s, and $-\infty$ if no such Ψ-set exists.

$$Table_u[\mathrm{s}] \overset{df}{=} \begin{cases} -\infty & \text{if } \Psi = \emptyset \\ optimum_{S \in \Psi}\{|\{v \in V(G_u) - B_u : states_S(v) \in M\}|\} & \text{otherwise} \end{cases}$$

Table Initialization

A leaf u of T is a Primitive node and G_u is the graph $G[B_u]$, where $B_u = \{w_1, ..., w_{k+1}\}$. Following the above definition we initialize $Table_u$ as follows

$$\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] = -\infty$$

$$\forall S \subseteq B : Table_u[\mathbf{s}] = 0 \text{ for } \mathbf{s} \text{ such that in } G[B_u] \; states_S(w_i) = s_i.$$

The complexity of this initialization for each leaf of T is $\mathcal{O}(|L|^{k+1} + 2^{k+2\log k})$.

Reduce Table
A Reduce node u of T has a single child a such that $B_u = \{w_1, ..., w_k\}$ and $B_a = \{w_1, ..., w_{k+1}\}$. We compute $Table_u$ based on correct $Table_a$ as follows

$$\forall \mathbf{s} \in I_k : Table_u[\mathbf{s}] = optimum\{Table_a[\mathbf{p}](+1 \text{ if } p_{k+1} \in M)\}$$

where the optimum (min or max) is taken over all $\mathbf{p} \in I_{k+1}$ such that $p_{k+1} \in F$ and $1 \leq i \leq k, p_i = s_i$.
Correctness of the operation follows by noting that G_a and G_u designate the same subgraph of G, and differ only by w_{k+1} not being a source in G_u. By definition, an entry of $Table_u$ optimizes over solutions where the state of w_{k+1} is one of the final states F and w_{k+1} contributes to the entry value if it has state in M. The complexity of this operation for each Reduce node of T is $\mathcal{O}(|L|^{k+1})$.

Join Tables
A Join node u of T has children a and b such that $B_u = B_a = \{w_1, ..., w_{k+1}\}$ and B_b is a k-subset of B_a, wlog let $B_b = \{w_1, ..., w_k\}$. Moreover, G_a and G_b share exactly the subgraph $G[B_b]$. We compute $Table_u$ based on correct tables of children as follows

$$\forall \mathbf{s} \in I_{k+1} : Table_u[\mathbf{s}] = optimum\{Table_a[\mathbf{p}] + Table_b[\mathbf{q}]\}$$

where the optimum (min or max) is taken over all compatible pairs $(\mathbf{p}, \mathbf{q})$ such that $\mathbf{p} \in I_{k+1}, \mathbf{q} \in I_k$, $p_{k+1} = s_{k+1}$. The pair compatibility is defined by $(p_i, q_i) \in \pi(i, \mathbf{s}), 1 \leq i \leq k$, with the function $\pi(i, \mathbf{s})$ defined next.
The function $\pi(i, \mathbf{s})$ gives the set of pairs of vertex states (p_i, q_i) which legally combine to form the vertex state s_i under the restriction of $\mathbf{s}$. This set of pairs depends on the number of source neighbors of w_i with σ-states. We define
$\alpha(i) = |\{w_j \in \{N_G(w_i) \cap B_a\} : s_j \text{ is a } \sigma\text{-state}\}|$ and
$\beta(i) = |\{w_j \in \{N_G(w_i) \cap B_b\} : s_j \text{ is a } \sigma\text{-state}\}|$.
We define π as follows

$$\pi(i, \mathbf{s}) \overset{df}{=} \begin{cases} \text{if } s_i = \rho_z \text{ then } \{(\rho_c, \rho_d) : c + d - \beta(i) = z\} \\ \text{if } s_i = \sigma_z \text{ then } \{(\sigma_c, \sigma_d) : c + d - \beta(i) = z\} \\ \text{if } s_i = \rho_{\geq z} \text{ then } \{(\rho_c, \rho_d)\} \cup \{(\rho_c, \rho_{\geq z})\} \cup \{(\rho_{\geq z}, \rho_d)\} \cup (\rho_{\geq z}, \rho_{\geq z}) \\ \text{if } s_i = \sigma_{\geq z} \text{ then } \{(\sigma_c, \sigma_d)\} \cup \{(\sigma_c, \sigma_{\geq z})\} \cup \{(\sigma_{\geq z}, \sigma_d)\} \cup (\sigma_{\geq z}, \sigma_{\geq z}) \end{cases}$$

where c and d are integer subscripts of legal states L, obeying the bounds imposed by $\alpha(i)$ and $\beta(i)$, respectively.

To show correctness of the computation, consider any $\mathbf{s} = s_1, ..., s_{k+1}$ and let $S \subseteq V(G_u)$ be a Ψ-set respecting G_u and $\mathbf{s}$ that optimizes the number of vertices among $V(G_u) - B_u$ having state in M. Let $S_a = V(G_a) \cap S$ and $S_b = V(G_b) \cap S$ and let $\mathbf{p} = p_1, ..., p_{k+1}$ (similarly $\mathbf{q} = q_1, ..., q_k$) be such that in G_a (respectively G_b) we have $state_{S_a}(w_i) = p_i$ (respectively $state_{S_b}(w_i) = q_i$). We want to show that the pair $(\mathbf{p}, \mathbf{q})$ is considered in the update of $Table_u[\mathbf{s}]$. Since $N_{G_u}(w_{k+1}) = N_{G_a}(w_{k+1})$ we have $p_{k+1} = s_{k+1}$. Next we show that $(p_i, q_i) \in \pi(i, \mathbf{s})$ for any $w_i, 1 \leq i \leq k$. Since S_a and S_b are subsets of S, if s_i is a σ-state then p_i, q_i are σ-states as well (and similarly for ρ-states).
Assume $s_i = \rho_z$ so that $z = |N_G(w_i) \cap S|$. Let $|N_{G_a}(w_i) \cap S_a| = c$, $|N_{G_b}(w_i) \cap S_b| = d$, so that $(p_i, q_i) = (\rho_c, \rho_d)$. Since $B_b \subseteq B_a$ we have $S_a \cap B_b = S_b \cap B_b = S \cap B_b$ and thus $|N_{G_a}(w_i) \cap S_a \cap B_b| = |N_{G_b}(w_i) \cap S_b \cap B_b| = |N_{G_u}(w_i) \cap S \cap B_b| = \beta(i)$, or $z = |N_{G_u}(w_i) \cap S| = (c - \beta(i)) + (d - \beta(i)) + \beta(i) = c + d - \beta(i)$. We conclude $(p_i, q_i) \in \pi(i, \mathbf{s})$. The remaining cases where s_i equal to $\sigma_z, \sigma_{\geq z}$ or $\rho_{\geq z}$ can be argued similarly.
To complete the proof of correctness we must show that Join-Tables does not consider too many entries. If $(\mathbf{p}, \mathbf{q}) \in \pi(i, \mathbf{s})$ and S_a is a Ψ-set with respect to G_a and $\mathbf{p}$ while S_b is a Ψ-set with respect to G_b and $\mathbf{q}$ then $S = S_a \cup S_b$ is itself a Ψ-set with respect to G_u and $\mathbf{s}$. This since B_b forms a separator of G_u and the definition of π contains the restriction $c + d - \beta(i) = z$ (or $c + d - \beta(i) \geq z$) on the number of selected neighbors each vertex in B_b has. We conclude that Join-Tables is correct.
For each Join node of T the complexity of Join-Tables is $\mathcal{O}(|L|^{2k+1})$ since any pair of entries from tables of children is considered at most once. Many of these pairs are in fact not considered at all and a refined analysis gives the complexity $\Theta(\Sigma_{\mathbf{s} \in I_k}(\Pi_{1 \leq i \leq k}|\pi(i, \mathbf{s})|))$ with the upper bound $\mathcal{O}(max_{i,\mathbf{s}}\{|\pi(i, \mathbf{s})|\}^k |L|^{k+1})$, which depends also on the particular legal states L of the problem.

Optimize Root Table

Let r be the root of T with $B_r = \{w_1, ..., w_k\}$. We compute $optM[F](G)$ based on correct $Table_r$ as follows

$$optM[F](G) = optimum\{Table_r[\mathbf{s}] + |\{w_i \in B_r : s_i \in M\}|\}$$

where the optimum (min or max) is taken over $\mathbf{s} \in I_k$ such that $s_i \in F, 1 \leq i \leq k$.
Correctness of this optimization follows from the definition of Table entries and the fact that G_r is the graph G with sources B_r.
The complexity of Table optimization at the root of T is $\mathcal{O}(|L|^{k+1})$.

Overall correctness and complexity

Correctness of the algorithms follows from a simple induction on the parse tree T. As noted in section 2, T has $n - k$ Primitive nodes, $n - k$ Reduce nodes and $n - k - 1$

Join nodes. The algorithms execute a single respective operation at each of these nodes, finds the parse tree T and performs Table Optimization at the root. The total time complexity becomes $T(n,k,L) = \mathcal{O}(n|L|^{2k+1})$, with Join Tables being the most expensive operation.

Theorem 2 *Algorithm-optM[F], with L the set of legal states with respect to F, computes $optM[F](G)$ and has time complexity $T(n,k,L) = \mathcal{O}(n|L|^{2k+1})$.*

Corollary 1 *For any problem $optM[F]$ derived from Table 2, Algorithm-optM[F] has time complexity $T(n,k) = \mathcal{O}(n2^{4k})$.*

The corollary follows since any problem derived from Table 2 has $|L| \leq 4$. Using the refined complexity analysis of Join-Tables we can get improvements on the overall complexity, the problem Maximum Independent Set achieving complexity $\mathcal{O}(n2^{k+2\log k})$.

5 Extensions

Our technique applies to a number of more general problems, as follows.

Search problems To construct an $[F]$-set of G optimizing the problem parameter we add pointers from each table entry to the table entries of children achieving the optimal value.

Weighted problems For weighted versions of the above problems, Table entries reflect optimization over the sums of weights of vertices and we need only modify the operations Table Reduce and Table Optimization. The Reduce operation adds the weight of the reduced vertex, when its state is in M, rather than incrementing the optimum sum by one. The Root operation, with the domain of optimization unchanged, becomes

$$optM[F](G) = optimum\{Table_r[\mathrm{s}] + \Sigma weight(w_i) : w_i \in B_r \wedge s_i \in M\}$$

Digraph problems For the directed graph versions of these problems we define $IN_G(v) = \{u :< u, v >\in Arcs\mathrm{G}\}$ and use $IN_G(v)$, as opposed to $N_G(v)$, in the definition of $states_S(v)$, the state of vertex v with respect to a selected set $S \subseteq V(G)$. The only change in the algorithm is for the definition of $\alpha(i), \beta(i)$ in Join-Tables that will use IN_G as well.

Maximal and minimal sets S is a *maximal* (minimal) $[F]$-set if no vertex can be removed from (added to) S such that the resulting set is still an $[F]$-set. Based on a

hand derived algorithm optimizing over all vertex subsets satisfying some property, [6] gives an automatic procedure constructing an algorithm optimizing over maximal (or minimal) vertex subsets satisfying the same property. This includes an application of Myhill-Nerode finite state automata minimization techniques to minimize the resulting number of separator states. For more on the connection with finite state automata, see also [1]. Unfortunately, when the original algorithm contains $|L|^k$ separator states, this automatic technique involves simplification of a table with $|L|^k 2^{|L|^k}$ separator states, and quickly becomes infeasible for increasing values of k.
To account for problems which optimize over maximal (or minimal) $[F]$-sets we need more elaborate changes. We now sketch a general procedure constructing a set of final vertex states $Fmax$ to identify maximal $[F]$-sets. Define
$A = \{\rho_i : \rho_i \in F \wedge \sigma_i \notin F\}$ and
$B = \{\rho_i : \rho_i \in F \wedge \rho_{i+1} \notin F\} \cup \{\sigma_i : \sigma_i \in F \wedge \sigma_{i+1} \notin F\}$.
S is a maximal $[F]$-set if and only if S is an $[F]$-set and $\forall v \in V(G) - S$ either $states_S(v) \in A$ or $\exists u \in N_G(v) : states_S(u) \in B$. Note that if A contains every ρ-state in F or if $B = F$ and the input graph has no isolated vertices then S is a maximal $[F]$-set precisely when S is an $[F]$-set. To identify maximal $[F]$-sets by vertex states we distinguish certain vertices with state in B to be a *mate* of neighboring vertices in $V(G) - S$ which are in turn distinguished to have a mate as neighbor. These two types of vertices will have states labelled *is* and *has* (a mate), respectively. The set of final states $Fmax$ for maximal $[F]$-sets will contain (i) any σ-state in F, (ii) any ρ-state in A, (iii) any state in B with the added label *is*, and (iv) any ρ-state in F with the added label *has*.
To design an algorithm solving a problem optimizing over maximal $[F]$-sets we use (a possibly hand-refined version of) $Fmax$ as starting point, find a corresponding set of legal vertex states and fill in details of Table Initialization, Reduce, Join and Root Optimization.
A construction in the same spirit will likewise give final vertex states for minimal $[F]$-sets. As an example, S is a minimal dominating set if it is a $[\rho_1^{is}, \rho_{\geq 1}, \sigma_0, \sigma_{\geq 0}^{has}]$-set.

Other problems The *irredundance* number of a graph is the minimum size of a maximal irredundant set. An explicit linear time algorithm computing the irredundance number of a tree has been given [6], and since irredundance is expressible as a linear EMSOL extremum problem [3] there exist linear time solution algorithms on partial k-trees, for any fixed k. We believe a more **practical** algorithm computing this parameter on a partial k-tree can be hand derived by the methods discussed in this paper. In our notation, S is an irredundant set if it is a $[\rho_0, \rho_1^{is}, \rho_{\geq 1}, \sigma_0, \sigma_{\geq 0}^{has}]$-set, and the close relation with minimal dominating sets is obvious. A set of vertex states for maximal irredundant sets can be constructed by a more complicated version of the procedure given above.

We have also applied our technique to problems for which we could not find a vertex state set with size independent of k. As an example, we mention Partition into Perfect Matchings (PPM):

For a graph G, find the minimum value of p for which there is a partition $V_1, V_2, ..., V_p$ of $V(G)$ such that $G[V_i], 1 \leq i \leq p$ has vertices of degree one only.

Below, we sketch a linear time algorithm for PPM on graphs of bounded treewidth (both [8] and [3] show only the existence of polynomial time algorithms). We observe that equivalent solutions to subproblems must induce identical partitions on the separator. We classify solutions by whether a given separator vertex has a mate among the other separator vertices, a mate among the reduced vertices, or no mate yet. Based on this we define a set of separator states and implementations of Initialize, Reduce, Join and Root-Optimize operations that give a linear time solution algorithm. The number of separator states as described here is $|I_k| = 3^k B(k)$, where $B(k)$ is the kth Bell number and the algorithm, although linear in n, is not very practical for increasing values of k.

References

[1] K.Abrahamson and M.Fellows, Finite automata, bounded treewidth and well-quasiordering, manuscript (1992).

[2] S.Arnborg, S.Hedetniemi and A.Proskurowski (editors) *Algorithms on graphs with bounded treewidth*, Special issue of *Discrete Applied Mathematics.*

[3] S.Arnborg, J.Lagergren and D.Seese, Easy problems for tree-decomposable graphs, *J. of Algorithms 12(1991) 308-340.*

[4] S. Arnborg and A. Proskurowski, Characterization and recognition of partial 3-trees, *SIAM J. Alg. and Discr. Methods 7 (1986) 305-314.*

[5] S.Arnborg and A.Proskurowski, Linear time algorithms for NP-hard problems on graphs embedded in k-trees, *Discr. Appl. Math. 23 (1989) 11-24.*

[6] M.W.Bern, E.L.Lawler and A.L.Wong, Linear-time computation of optimal subgraphs of decomposable graphs, *J. of Algorithms 8(1987) 216-235.*

[7] H.L.Bodlaender, Dynamic programming on graphs with bounded treewidth, *Proceedings ICALP 88, LNCS vol.317 (1988) 105-119.*

[8] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, manuscript (1992).

[9] J.A.Bondy and U.S.R.Murty, *Graph theory with applications*, 1976.

[10] R.B.Borie, R.G.Parker and C.A.Tovey, Automatic generation of linear algorithms from predicate calculus descriptions of problems on recurive constructed graph families, manuscript (1988).

[11] E.J.Cockayne, B.L.Hartnell, S.T.Hedetniemi and R.Laskar, Perfect domination in graphs, manuscript (1992), to appear in Special issue of JCISS.

[12] B.Courcelle, The monadic second-order logic of graphs I: Recognizable sets of finite graphs, *Information and Computation, 85: (1990)12-75.*

[13] M.Garey and D.Johnson, "Computers and Intractability", Freeman, San Fransisco, 1979.

[14] D.Grinstead and P.Slater, A recurrence template for several parameters in series-parallel graphs, manuscript (1992).

[15] J. van Leeuwen, Graph Algorithms, in *Handbook of Theoretical Computer Science vol. A* , Elsevier, Amsterdam, (1990) pg.550.

[16] J.Matoušek and R.Thomas, Algorithms finding tree-decompositions of graphs, *J. of Algorithms*,12 (1991) 1-22.

[17] A.Proskurowski and M.Syslo, Efficient computations in tree-like graphs, in *Computing Suppl. 7,(1990) 1-15.*

[18] N. Robertson and P.D. Seymour, Graph minors II: algorithmic aspects of tree-width, *J. of Algorithms 7 (1986) 309-322.*

[19] D.Sanders, On linear recognition of tree-width at most four, manuscript (1992).

[20] K.Takamizawa, T.Nishizeki and N.Saito, Linear-time computability of combinatorial problems on series-parallel graphs, *J. ACM 29(1982) 623-641.*

[21] J.A.Telle, Characterization of domination-type problems and their complexity, in preparation.

[22] J.A.Telle and A.Proskurowski, Efficient sets in partial k-trees, to appear in *Domination in graphs*, Special volume of *Discrete Mathematics.*

[23] T.Wimer, Linear time algorithms on k-terminal graphs. Ph.D. thesis, Clemson University, South Carolina, (1988).