Reduction Algorithms for Graphs of Small Treewidth[*]

Hans L. Bodlaender
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
e-mail: hansb@cs.ruu.nl

Babette de Fluiter
Centre for Quantitative Methods
P.O. Box 414, 5600 AK Eindhoven, the Netherlands
e-mail: deFluiter@cqm.nl

Abstract

This paper presents a number of new ideas and results on graph reduction applied to graphs of bounded treewidth. Arnborg et al. [2] have shown that many decision problems on graphs can be solved in linear time on graphs of bounded treewidth, by using a finite set of reduction rules. These algorithms can be used to solve problems on graphs of bounded treewidth without the need to first obtain a tree decomposition of the input graph. We show that the reduction method can be extended to solve the construction variants of many decision problems on graphs of bounded treewidth, including all problems definable in monadic second order logic.

We also show that a variant of the reduction algorithms presented in [2] can be used to solve (constructive) *optimization* problems in $O(n)$ time. For example, optimization and construction variants of INDEPENDENT SET and HAMILTONIAN COMPLETION NUMBER can be solved in this way on graphs of small treewidth.

Additionally we show that the results of [8] can be applied to our reduction algorithms, which results in parallel reduction algorithms that use $O(n)$ operations and $O(\log n \log^* n)$ time on an EREW PRAM, or $O(\log n)$ time on a CRCW PRAM.

1 Introduction

In this paper we discuss *reduction algorithms* for decision and optimization problems. A reduction algorithm is based on a finite set of *reduction rules* and a finite set of graphs. Each reduction rule describes a way to modify a graph locally. The original idea of a reduction algorithm is to solve a decision problem by repeatedly applying reduction rules on the input graph until no more rule can be applied. If the resulting graph is in the finite set of graphs, then the algorithm returns true, otherwise it returns false.

[*]This research was carried out while the second author was working at the Department of Computer Science of Utrecht University, with support by the Foundation for Computer Science (S.I.O.N) of the Netherlands Organization for Scientific Research (N.W.O.). Parts of this research have been published in preliminary form in [7, 5].

The idea of reduction algorithms originates from Duffin's [12] characterization of series-parallel graphs: a multigraph is series-parallel if and only if it can be reduced to a single edge by applying a sequence of *series* and *parallel* reductions. In [18] it was shown how a reduction algorithm based on this set of reduction rules can be implemented in linear time, and hence series-parallel graphs can be recognized in linear time.

Arnborg and Proskurowski [4] extended these ideas, and obtained reduction rules that characterize the graphs of treewidth at most two or three, and, amongst others, showed that these reduction rules can be used to recognize graphs of treewidth at most three in $O(n^3)$ time. In [17] it is shown that with a slightly different set of reduction rules graphs of treewidth at most three can be recognized in linear time. Additionally, a tree decomposition of minimum width can be constructed in linear time if the input graph has treewidth at most three.

A much more general approach is taken in [2]: a set of conditions is given that must hold for a set of reduction rules to ensure that the reduction algorithm works correctly. It is also shown that for all finite state decision problems on graphs of bounded treewidth, there is a set of reduction rules for which these conditions hold, and that the algorithm based on such a set of reduction rules takes $O(n)$ time (but more than linear space). The finite state decision problems include all MS-definable decision problems. The results of [2] are stated in a general, algebraic setting.

Bodlaender and Hagerup [8] have shown that the sequential reduction algorithms of [2] and [5] can efficiently be parallelized, if some additional conditions hold for the set of reduction rules. Their reduction algorithm uses $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM, and $O(\log n)$ time with $O(n)$ operations and space on a CRCW PRAM. A sequential version of this algorithm gives a reduction algorithm which uses $O(n)$ time and space. They show that such sets of reduction rules can be found for all finite state decision problems, assuming yes-instances have bounded treewidth.

In this paper, we extend these results in two directions. We show that reduction algorithms can also be used to solve constructive versions of many problems, and we show that reduction algorithms can also be used to solve some optimization problems, still assuming bounded treewidth of yes-instances. We also discuss parallelizations of these algorithms.

Many decision problems have a constructive version, in which we are not only interested in whether a certain property holds for a given graph, but we are also interested in a *solution*, if the property holds. For example, in the constructive version of k-COLORABILITY we want to find a k-coloring of a given graph, if one exists. Ordinary reduction algorithms do not provide a possibility to construct solutions, but only decide upon membership in a class of graphs. In this paper we show how reduction algorithms can be adapted in such a way that solutions can be constructed, and we show that these algorithms run within the same time and resource bounds as the basic reduction algorithms (both sequentially and in parallel). We also show that for a number of graph problems on graphs of bounded treewidth, the technique can be used, including all MS-definable construction problems whose solution structure satisfies certain conditions.

Ordinary reduction algorithms (with the extension described above) can be used for (constructive) decision problems. In this paper, we extend the notion of reduction algorithms to (constructive) optimization problems: we introduce a new notion of reduction rules for optimization problems, called *reduction-counter rules*, and give a set of conditions which are necessary for a set of reduction-counter rules in order to make a reduction algorithm work correctly. This results in efficient reduction algorithms for (constructive) optimization problems which run within the same time and resource bounds as the original reduction algorithms, both sequentially and in parallel. For simple graphs of bounded treewidth this gives efficient algorithms for a number of optimization problems.

This paper is organized as follows. In Section 2 we discuss reduction algorithms for decision problems as introduced in [2], and reprove some results, but in a less algebraic setting. We also give a reduction algorithm which uses linear time and space, based on the ideas of [2] and of [8]. In Section 3 we extend the theory of reduction algorithms for decision problems to constructive reduction algorithms. In Sections 4 and 5 we extend the notion of reduction algorithms and constructive reduction algorithms to optimization problems. In Section 6, we discuss the parallel reduction algorithms of [8], and in Section 7, we mention some additional results.

For reasons of clarity we present the reduction algorithms in this paper for problems on connected graphs. In Section 7 we briefly discuss how to extend these results to graphs which are not necessarily connected.

2 Reduction Algorithms for Decision Problems

In this section we discuss the results of [2], and reprove some of these results, but in a more direct way, avoiding the algebraic setting from [2] — this facilitates our later extensions of the results. We start with definitions or reduction rules and reduction systems (Section 2.1). Then we give an efficient reduction algorithm based on a special type of reduction system (Section 2.2). Finally, we show that this reduction algorithm can be used to solve a large class of decision problems on graphs of bounded treewidth (Section 2.3).

2.1 Reduction Systems

The graphs we consider are simple and do not contain self-loops, unless stated otherwise.

A *graph property* is a function P which maps each graph to the value true or false. We assume that isomorphic graphs are mapped to the same value. We say that P holds for graph G or $P(G)$ holds, if $P(G) = \text{true}$. A graph property P corresponds directly to a decision problem: given a graph G, does P hold for G? An algorithm decides a property P if it solves the corresponding decision problem. A property is *effectively decidable* if an algorithm is *known* that decides the property.

Definition 2.1 (Terminal Graph). *A* terminal graph *G is a triple (V,E,X) with (V,E) a simple graph, and $X \subseteq V$ an ordered subset of $l \geq 0$ vertices. We denote X by $\langle x_1, \ldots, x_l \rangle$. Vertices in X are called* terminals *or* terminal vertices. *Vertices in $V \Leftrightarrow X$ are called* inner vertices.

The graphs G and H depicted in Figure 1 are examples of terminal graphs.

A terminal graph with l terminals $(l \geq 0)$ is also called an l-terminal graph. Let $G = (V,E,X)$ be an l-terminal graph, $l \geq 0$, with $X = \langle x_1, \ldots, x_l \rangle$. For each i, $1 \leq i \leq l$, we call x_i the ith terminal of G. A terminal graph (V,E,X) is said to be *open* if there are no edges between its terminals.

Definition 2.2. *The operation $\oplus$ maps two terminal graphs G and H with the same number l of terminals to a simple graph $G \oplus H$, by taking the disjoint union of G and H, then identifying for $i = 1, \ldots, l$, the ith terminal of G with the ith terminal of H, and removing multiple edges.*

For an example of the $\oplus$-operation, see Figure 1. Note that the result of an $\oplus$ operation is a simple graph, and not a terminal graph.

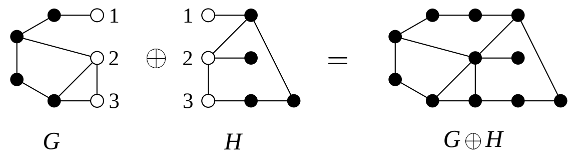

Figure 1: Example of operation $\oplus$ applied to two three-terminal graphs.

Two terminal graphs $(V_1, E_1, \langle x_1, \cdots, x_k \rangle)$ and $(V_2, E_2, \langle y_1, \cdots, y_l \rangle)$ are said to be *isomorphic* if $k = l$ and there is an isomorphism from (V_1, E_1) to (V_2, E_2) which maps x_i to y_i for each i, $1 \leq i \leq k$.

Definition 2.3 (Reduction Rule). *A* reduction rule *r is an ordered pair (H_1, H_2), where H_1 and H_2 are l-terminal graphs for some $l \geq 0$.*

A match *to reduction rule $r = (H_1, H_2)$ in graph G is a terminal graph G_1 which is isomorphic to H_1, such that there is a terminal graph G_2 with $G = G_1 \oplus G_2$.*

If G contains a match to r, then an application *of r to G is an operation that replaces G by a graph G', such that there are terminal graphs G_1, G_2 and G_3, with G_1 isomorphic to H_1, G_2 isomorphic to H_2, and $G = G_1 \oplus G_3$, $G' = G_2 \oplus G_3$. We also say that, in G, G_1 is replaced by G_2. An application of a reduction rule is also called a* reduction.

Figure 2 shows an example of a reduction rule r, and an application of r to a graph G. We depict a reduction rule (H_1, H_2) by the two graphs H_1 and H_2 with an arrow from H_1 to H_2. Given a reduction rule $r = (H_1, H_2)$, we call H_1 the left-hand side of r, and H_2 the right-hand side of r.

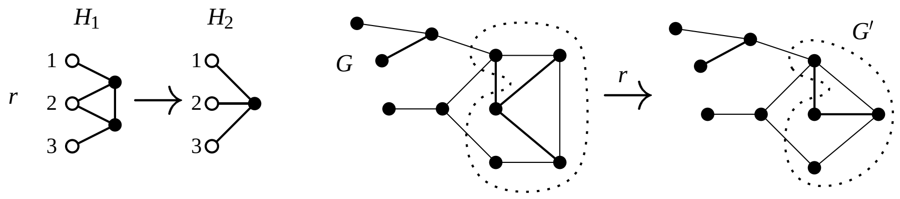

Figure 2: An example of a reduction rule $r = (H_1, H_2)$, and an application of r to a graph G, resulting in graph G'. The dotted lines in G and G' denote the parts of G and G' that are involved in the reduction.

Let G be a graph and $r = (H_1, H_2)$ a reduction rule. If G contains a match G_1 to r, then an application of r to G which replaces G_1 by a terminal graph isomorphic to H_2 is called a reduction corresponding to the match G_1.

If there is an application of rule r to graph G which results in a graph G', then we write $G \xrightarrow{r} G'$. Let $\mathcal{R}$ be a set of reduction rules. For two graphs G and G', we write $G \xrightarrow{\mathcal{R}} G'$ if there exists an $r \in \mathcal{R}$ with $G \xrightarrow{r} G'$. We say G contains a match G_1 if there is an $r \in \mathcal{R}$ such that G_1 is a match to r in G. If G contains no match, then we say that G is *irreducible* (for $\mathcal{R}$).

The following conditions are useful for a set of reduction rules in order to get a characterization of a graph property P.

Definition 2.4. *Let P be a graph property and $\mathcal{R}$ a set of reduction rules.*

- $\mathcal{R}$ *is* safe *for P if, whenever $G \xrightarrow{\mathcal{R}} G'$, then $P(G) \Leftrightarrow P(G')$.*
- $\mathcal{R}$ *is* complete *for P if the set $\mathcal{I}$ of irreducible graphs for which P holds is finite.*
- $\mathcal{R}$ *is* decreasing *if, whenever $G \xrightarrow{\mathcal{R}} G'$, then G' contains fewer vertices than G.*

Definition 2.5 (Reduction System). *A* reduction system *for a graph property P is a pair $(\mathcal{R}, \mathcal{I})$, with $\mathcal{R}$ a finite set of reduction rules which is safe, complete and decreasing for P, and $\mathcal{I}$ the set of irreducible graphs for which P holds.*

A reduction system $(\mathcal{R}, \mathcal{I})$ for a property P gives a complete characterization of P: $P(G)$ holds for a graph G if and only if any sequence of reductions from $\mathcal{R}$ on G leads to a graph G' which belongs to $\mathcal{I}$ (i.e. is isomorphic to a graph in $\mathcal{I}$).

2.2 An Efficient Reduction Algorithm

A reduction system $(\mathcal{R}, \mathcal{I})$ for a property P corresponds to a polynomial time algorithm that decides whether property P holds for a given graph G: repeat applying rules from $\mathcal{R}$ starting with the input graph, until no rule from $\mathcal{R}$ can be applied anymore. If the resulting graph belongs to the set $\mathcal{I}$, then P holds for the input graph, otherwise, it does not. The number of reductions that has to be performed is at most n, since each reduction reduces the number of vertices by at least one. In order to obtain a linear time reduction algorithm, we define a special type of reduction system $(\mathcal{R}, \mathcal{I})$ which has the property that for any graph G for which $P(G)$ holds, either G belongs to $\mathcal{I}$, or G contains a match which can be found in an efficient way. We consider the method used in [8], called the *bounded adjacency list search method*. (In [8] this method is used to obtain an efficient parallel algorithm; we give an efficient sequential version of this parallel algorithm in this section.)

Definition 2.6. *Let d be a positive integer. Let G be a graph given by some adjacency list representation and let G_1 be an l-terminal graph. We say G_1 is d-*discoverable *in G if*

1. *G_1 is open and connected, and the maximum degree of any vertex in G_1 is at most d,*
2. *there is an l-terminal graph G_2 such that $G = G_1 \oplus G_2$, and*
3. *G_1 contains an inner vertex v such that for all vertices $w \in V(G_1)$ there is a walk W in G_1 with $W = (u_1, u_2, \ldots, u_s)$, $v = u_1$, $w = u_s$, and for each i, $2 \leq i \leq s \Leftrightarrow 1$, in the adjacency list of u_i in G, the edges $\{u_{i-1}, u_i\}$ and $\{u_i, u_{i+1}\}$ have distance at most d.*

Let G be a graph, d a positive integer, and let G_1 be a d-discoverable terminal graph in G. It can be seen that there is a walk from any inner vertex w to any other vertex w' in G_1 in which two subsequent edges have distance at most d in the adjacency list of their common vertex, and each edge occurs at most twice. This, and the fact that each edge in an open terminal graph G_1 is incident with an inner vertex (which has degree at most d) implies the following result.

Lemma 2.1. *If a terminal graph G_1 is d-discoverable in a graph G for some fixed $d \geq 1$, then for any inner vertex v of G_1, all vertices and edges of G_1 can be found from v in an amount of time that only depends on the integer d and the size of G_1, but not on the size of the graph G.*

Definition 2.7 (Special Reduction System). *Let P be a graph property and $(\mathcal{R}, \mathcal{I})$ a reduction system for P. Let n_{max} be the maximum number of vertices in any left-hand side of a rule $r \in \mathcal{R}$. $(\mathcal{R}, \mathcal{I})$ is a* special reduction system *for P if we know positive integers n_{min} and d, $n_{min} \leq n_{max} \leq d$, such that the following conditions hold.*

1. *For each reduction rule $(H_1, H_2) \in \mathcal{R}$, H_1 and H_2 are open and connected.*
2. *For each connected graph G and each adjacency list representation of G, if $P(G)$ holds and G has at least n_{min} vertices, then G contains a d-discoverable match.*

As a simple example of a special reduction system, consider the graph property P, where $P(G)$ holds if and only if G is a two-colorable cycle. Let $\mathcal{R}$ contain the one reduction rule depicted in Figure 3, and let $\mathcal{I}$ be the set containing just the cycle on four vertices (see also Figure 3). It can easily be seen that $(\mathcal{R}, \mathcal{I})$ is a special reduction system for P ($d = n_{max} = n_{min} = 5$).

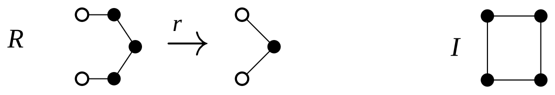

Figure 3: A reduction system for the property that a graph is a two-colorable cycle.

In Section 2.3, we show that one can find special reduction systems for many problems on graphs of bounded treewidth.

Theorem 2.1. *Let P be a graph property. If we have a special reduction system for P, then we have an algorithm which decides P in $O(n)$ time and $O(n)$ space on connected graphs.*

Suppose we have a special reduction system for property P. Our algorithm finds d-discoverable matches and executes the corresponding reductions, until there are no more d-discoverable matches. If the resulting graph is in $\mathcal{I}$, then P holds for the input graph, and true is returned. Otherwise, false is returned. The algorithm is a simplified sequential simulation of the parallel algorithm given in [8]. It resembles the algorithm of [2], but uses $O(n)$ space, whereas the algorithm of [2] uses $\Omega(n^p)$ space, where p equals the maximum number of terminal vertices in any reduction rule.

We now give the complete algorithm, given the special reduction system $(\mathcal{R}, \mathcal{I})$ and the integers n_{min} and d.

```
Algorithm Reduce($G$)
Input: Connected graph $G$
Output: $P(G)$
1.  $n_{max} \leftarrow \max\{|V(H)| \mid H$ is left-hand side of some $r \in \mathcal{R}\}$
2.  $S \leftarrow \{v \in V(G) \mid \deg(v) \leq d\}$
3.  while $S \neq \emptyset$
4.      do take $v \in S$
5.          if $v$ is inner vertex of a $d$-discoverable match $G_1$ to a rule $r \in \mathcal{R}$
6.            then apply $r$ to $G$:
7.                   let $G_2$ be a new terminal graph isomorphic to $H_2$, such that $G_1$ and $G_2$ have
                     the same set of terminals
8.                   replace $G_1$ by $G_2$
9.                   $S \leftarrow S \Leftrightarrow \{v \in V(G_1) \mid v$ is inner vertex of $G_1\}$
10.                  $S \leftarrow S \cup \{v \in V(G_2) \mid \deg(v) \leq d\}$
11.                  for all terminals $x$ of $G_2$
12.                      do let $L$ denote adjacency list of $x$
13.                          for all $\{x, w\} \in L$ for which $L$ changed within distance $d$
14.                              do if $\deg(w) \leq d$ then $S \leftarrow S \cup \{w\}$
15.         $S \leftarrow S \Leftrightarrow \{v\}$
16. if $G \in \mathcal{I}$ then return true else return false
```

We first show that the algorithm is correct.

Lemma 2.2. *Algorithm Reduce correctly recognizes connected graphs for which a property P holds, given a special reduction system $(\mathcal{R}, \mathcal{I})$ for P.*

Proof. Suppose the input graph is connected. Now, one can establish three invariants for the main loop of the algorithm (G is the graph the algorithm 'works with'): G is connected; $P(G)$ holds if and only if P holds for the input graph; for each d-discoverable match G_1 in G, there is a vertex $w \in S$ which is an inner vertex of G_1. Correctness of the algorithm follows from these invariants, whose proof we leave to the reader (see [11] for full details). $\square$

Consider the time and space complexity of the algorithm.

Lemma 2.3. *Algorithm Reduce uses $O(n)$ time and space.*

Proof. We first show that the main loop of the algorithm is iterated $O(n)$ times. We do this by showing that the number of times a vertex is added to S is $O(n)$. Initially, in line 2, S contains $O(n)$ vertices. In the main loop, there are only vertices added to S if a reduction takes place. Since at most n reductions take place, and after each reduction, at most a constant number of vertices is added to S, this means that the total number of vertices added to S during the main loop is also $O(n)$. Since in each iteration of the main loop, at least one vertex is removed from S, this means that the main loop is executed $O(n)$ times.

Consider one iteration of the main loop. In line 5, a d-discoverable match in G that contains v as an inner vertex can be found in constant time, as we described. Furthermore, each reduction can be done in constant time. The loop in lines 11 – 14 can also be done in constant time: during the reduction, it is possible to store the places in the adjacency lists of the terminals where something changes, so that they can be easily found. Hence each iteration of the main loop takes $O(1)$ time, and the algorithm can be done in $O(n)$ time.

It is easy to see that the amount of space used by the algorithm is $O(n)$. □

This completes the proof of Theorem 2.1.

2.3 Decision Problems for Graphs of Bounded Treewidth

In this section, we show that algorithm Reduce can be used for a large class of graph properties on graphs of bounded treewidth.

Definition 2.8 (Tree Decomposition & Treewidth). *Let $G = (V,E)$ be a graph. A* tree decomposition *TD of G is a pair (T,X), where $T = (I,F)$ is a tree, and $X = \{X_i \mid i \in I\}$ is a family of subsets of V, one for each node (vertex) of T, such that*

- $\bigcup_{i\in I} X_i = V$,
- *for every edge $\{v,w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and*
- *for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of a tree decomposition $((I,F),\{X_i \mid i \in I\})$ is $\max_{i\in I} |X_i| \Leftrightarrow 1$. The treewidth of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.*

Definition 2.9. *Let P be a graph property, and l a non-negative integer. For l-terminal graphs G_1 and G_2, we define the equivalence relation $\sim_{P,l}$ as follows:*

$$G_1 \sim_{P,l} G_2 \quad \Leftrightarrow \quad \text{for all } l\text{-terminal graphs } H\text{: } P(G_1 \oplus H) \Leftrightarrow P(G_2 \oplus H).$$

Property P is of finite index *if for all $l \geq 0$, $\sim_{P,l}$ has finitely many equivalence classes.*

Note that a set $\mathcal{R}$ of reduction rules for a property P is safe if and only if for each reduction rule $(H_1,H_2) \in \mathcal{R}$, $H_1 \sim_{P,l} H_2$, for l the number of terminals of H_1 and H_2.

An equivalence relation $\sim'$ is a *refinement* of an equivalence relation $\sim$ if each equivalence class of $\sim'$ is a subset of an equivalence class of $\sim$. Clearly, if $\sim'$ is finite, then so is $\sim$.

The following result is well-known.

Lemma 2.4 [9, 13]. *Let P_1 and P_2 be graph properties of finite index. Let Q_1 and Q_2 be graph properties defined as follows: for each graph G, $Q_1(G) = P_1(G) \wedge P_2(G)$, and $Q_2(G) = P_1(G) \vee P_2(G)$. Then Q_1 and Q_2 are also of finite index.*

For each integer $k \geq 1$, let TW_k be the graph property defined as follows: for each graph G, $TW_k(G)$ holds if and only if $\mathrm{tw}(G) \leq k$.

Lemma 2.5 [3, 16]. *For each fixed $k \geq 1$, TW_k is of finite index, and for each $l \geq 0$, there is a finite, effectively decidable refinement of $\sim_{TW_k,l}$.*

For a property P and an integer k, we define the property P_k as $P_k(G) = P(G) \wedge TW_k(G)$. It follows from Lemmas 2.4 and 2.5 that for each fixed $k \geq 1$, if P is of finite index, then so is P_k, and furthermore, if we have a refinement $\sim_l$ of $\sim_{P,l}$ which is effectively decidable, then we have a refinement $\sim'_l$ of $\sim_{P_k,l}$ which is effectively decidable.

Finite index corresponds to 'finite state': there exists a linear time algorithm that decides finite index properties on graphs, given their tree decomposition of bounded treewidth. Moreover, this algorithm is of a special, well-described structure [10, 9, 1]. The disadvantage of this algorithm is that a tree decomposition of the input graph is needed. Although for each fixed k, there is a linear time sequential algorithm which, given a graph G, checks if $\mathrm{tw}(G) \leq k$, and if so, computes a minimum width tree decomposition of G [6], this algorithm is not very practical, due to the large constant factors involved. With reduction algorithms, one does not need to build a tree decomposition first.

Lemma 2.6 [8]. *Let k and n_{min} be positive integers. There are integers d and n_{max}, $2(n_{min} \Leftrightarrow 1) \leq n_{max} \leq d$, and a constant $c > 0$, such that in each connected graph G of treewidth at most k, if $n \geq n_{min}$, then G contains at least $\lceil cn \rceil$ d-discoverable open and connected l-terminal graphs H with $l \leq 2(k+1)$ and $n_{min} \leq |V(H)| \leq n_{max}$.*

The following theorem has originally been proved in [2] for a slightly different kind of special reduction system. In [8] the proof was adapted for the special reduction system as defined here. Using the techniques from [2, 8] we give a proof with some details in a different form.

Theorem 2.2. *Let P a graph property and suppose P is of finite index. For each integer $k \geq 1$, there exists a special reduction system $(\mathcal{R}, \mathcal{I})$ for P_k.*

If P is also effectively decidable, and there is an equivalence relation $\sim_l$ for each $l \geq 0$ which is a finite refinement of $\sim_{P,l}$ and is effectively decidable, then such a system $(\mathcal{R}, \mathcal{I})$ can effectively be constructed.

Proof. Let $k \geq 1$. We first construct all right-hand sides of reduction rules. For every $l \leq 2(k+1)$ and every equivalence class C of $\sim_{P_k,l}$, do the following. If C contains open and connected l-terminal graphs with treewidth at most k, then choose a representing open and connected l-terminal graph $H_C \in C$ with treewidth at most k. Let n_{min} be one more than the maximum number of vertices of all chosen graphs H_C. Let d, n_{max} and c be as in Lemma 2.6.

Let $\mathcal{R}$ denote the set of reduction rules to be built. For all l with $0 \leq l \leq 2(k+1)$ and for all open connected l-terminal graphs H with at least n_{min} and at most n_{max} vertices, if we have a representative for the equivalence class C in which H is contained, then add the reduction rule (H, H_C) to $\mathcal{R}$. Note that if we do not have such a representative, then either H must have treewidth at least $k+1$, or H is not open and connected. In the first case there is no terminal graph G for which $P_k(H \oplus G)$ holds. In the latter case, H is not discoverable in any graph.

Let $\mathcal{I} = \{G \mid G \text{ is irreducible } \wedge P_k(G) \wedge G \text{ is connected}\}$.

It is easy to see that $\mathcal{R}$ is finite: there are finitely many l-terminal graphs with at most n_{max} vertices. Safeness of the resulting set $\mathcal{R}$ follows directly from the fact that each left- and right-hand side of a rule in $\mathcal{R}$ belong to the same equivalence class of the relation $\sim_{P_k,l}$.

Condition 1 of a special reduction system (Definition 2.7) clearly holds, and $\mathcal{R}$ is decreasing.

We now show that $\mathcal{R}$ is complete, i.e. that $|\mathcal{I}|$ is finite and that condition 2 of Definition 2.7 holds. Let G be a graph for which $P_k(G)$ holds. Note that $\mathrm{tw}(G) \leq k$. If G has at least n_{min} vertices, then, by Lemma 2.6, C contains at least $\lceil c|V(C)|\rceil \geq 1$ d-discoverable open l-terminal graphs H with $l \leq 2(k+1)$ and $n_{min} \leq |V(H)| \leq n_{max}$. Hence, by construction of the reduction system, G contains a d-discoverable match, so condition 2 holds.

Clearly, all graphs in $\mathcal{I}$ have less than n_{min} vertices, and hence $|\mathcal{I}|$ is finite. This completes the proof that $\mathcal{R}$ is complete, and hence that $(\mathcal{R},\mathcal{I})$ is a special reduction system.

The effective construction of the reduction system can be done using the results from [2] and [16]. □

From the proof of Theorem 2.2, we can also conclude the following.

Corollary 2.1. *Let P be a graph property, and for each $l \geq 0$, let $\sim_l$ be a refinement of $\sim_{P,l}$. Let $k \geq 1$. If $\sim_l$ is finite for each $l \geq 0$, then there is a special reduction system $(\mathcal{R},\mathcal{I})$ for P_k, such that for each $(H,H') \in \mathcal{R}$, $H \sim_l H'$. Moreover, if $\sim_l$ and P are effectively decidable, then such a system can effectively be constructed.*

Courcelle [10] has given a large class of graph properties which are of finite index, namely the class of properties that are definable in *Monadic Second Order Logic* or MSOL for graphs. MSOL for graphs $G = (V,E)$ consists of a language in which predicates can be built with

- the logic connectives $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and $\Leftrightarrow$ (with their usual meanings),
- individual variables which may be vertex variables (with domain V), edge variables (with domain E), vertex set variables (with domain $\mathcal{P}(V)$, the power set of V), and edge set variables (with domain $\mathcal{P}(E)$),
- the existential and universal quantifiers ranging over variables ($\exists$ and $\forall$, respectively), and
- the following binary relations:
  - $v \in W$, where v is a vertex variable and W a vertex set variable,
  - $e \in F$, where e is an edge variable and F an edge set variable,
  - 'v and w are adjacent in G', where v and w are vertex variables,
  - 'v is incident with e in G', where v is a vertex variable, and e an edge variable, and
  - equality for variables.

Graph properties that can be defined by an MSOL predicate are called *MS-definable* graph properties. In [10] is was shown that MS-definable graph properties are of finite index. There are many (even NP-complete) decision problems which are MS-definable (i.e. the corresponding graph properties are MS-definable). These include HAMILTONIAN CIRCUIT and (for fixed k) k-COLORABILITY (see e.g. [3] for a list). Theorem 2.2 now immediately implies the following result.

Corollary 2.2. *Let P be a graph property which is MS-definable. For each integer $k \geq 1$, there is a linear time algorithm which decides P_k on connected graphs without using a tree decomposition of the input graph. Moreover, such an algorithm can be automatically constructed from an MSOL predicate for P.*

3 Reduction Algorithms for Construction Problems

In this section we extend the results discussed in the previous section to construction problems: we extend the reduction algorithms to *constructive* reduction algorithms, which can be used to construct solutions for decision problems.

The basic idea of a constructive reduction algorithm is the following. The algorithm consists of two parts. In the first part, an ordinary reduction algorithm is applied. The reduced graph is then passed to the second part. In this part, a solution is constructed for the reduced graph, if it exists. After that, the reductions that are applied in part 1 are undone one by one, in reversed order, and each time a reduction is undone, the solution of the graph is adapted to a solution of the new graph. This results in a solution of the input graph.

In order to keep the running time and amount of resources for the second part within the same bounds as for first part, we must be able to efficiently construct a solution for the new graph from a solution of the old graph, after an undo-action is applied. Therefore, we require that the new solution can efficiently be constructed from the old solution.

In this section we start with definitions of a constructive reduction system and an extension of the efficient reduction algorithm presented in Section 2.2 to construction problems. After that, we show how this algorithm can be applied to solve a large class of construction problems on graphs of bounded treewidth.

3.1 Constructive Reduction Systems and Algorithms

Many graph properties are of the form

$$P(G) = \text{`there is an } S \in D(G) \text{ for which } Q(G,S) \text{ holds'},$$

where $D(G)$ is a *solution domain* (or shortly domain), which is some set depending on G, and Q is an *extended* graph property of G and S, i.e. $Q(G,S) \in \{\mathsf{true}, \mathsf{false}\}$ for all graphs G and all $S \in D(G)$. An $S \in D(G)$ for which $Q(G,S)$ holds is called a *solution* for G. For example, for the perfect matching problem on a graph G, $D(G)$ can be $\mathcal{P}(E)$, the power set of E, and for $S \in D(G)$, $Q(G,S)$ holds if and only if every vertex in G is end point of exactly one edge in S. Hence S is a solution for G if S is a perfect matching of G.

If a graph property is of the form $P(G) =$ 'there is an $S \in D(G)$ for which $Q(G,S)$ holds', then we call P a *construction property* defined by the pair (D,Q).

In this section, we introduce constructive reduction algorithms which, for a construction property P defined by (D,Q), do not only decide P, but if P holds for an input graph G, also construct an $S \in D(G)$ for which $Q(G,S)$ holds. To this end, we generalize the notion of reduction systems.

Definition 3.1 (Constructive Reduction System). *Let P be a construction property defined by (D,Q). A* constructive reduction system *for P is a quadruple $(\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$, where*

- *$(\mathcal{R}, \mathcal{I})$ is a reduction system for P,*
- *$A_{\mathcal{R}}$ is an algorithm which, given*
    - *a reduction rule $r = (H_1, H_2) \in \mathcal{R}$,*

  - *two terminal graphs G_1 and G_2, such that G_1 is a isomorphic to H_1 and G_2 is isomorphic to H_2,*
  - *a graph G with $G = G_2 \oplus H$ for some H, and*
  - *an $S \in D(G)$ for which $Q(G,S)$ holds,*

  computes an $S' \in G_1 \oplus H$ such that $Q(G_1 \oplus H, S')$ holds,
- *$A_{\mathcal{I}}$ is an algorithm which, given a graph G which is isomorphic to some $H \in \mathcal{I}$, computes an $S \in D(G)$ for which $Q(G,S)$ holds.*

Algorithm $A_{\mathcal{I}}$ in a constructive reduction system $(\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$ is used to construct an initial solution of the reduced graph G, if $G \in \mathcal{I}$. Algorithm $A_{\mathcal{R}}$ is used to reconstruct a solution, each time a reduction is undone on the graph.

As an example, consider the constructive version of the graph property P which holds for graphs G which are two-colorable cycles (see the example of Figure 3): we are looking for a two-coloring of the graph, if the graph is a two-colorable cycle. For each graph G, let $D(G)$ be the set of partitions (V_1, V_2) of $V(G)$, and for each $S \in D(G)$, let $Q(G,S)$ be true if and only if G is a cycle and S is a two-coloring of G.

We extend the reduction system for P given in Figure 3 to a constructive reduction system for P. Algorithm $A_{\mathcal{R}}$ uses a table: for reduction rule $r = (H_1, H_2) \in \mathcal{R}$, and each possible two-coloring of the terminal graph H_2, it gives a two-coloring of the terminal graph H_1 which is the same on the set of terminals. The contents of this table are depicted in part I of Figure 4 (symmetric cases are considered only once, hence there is only one two-coloring). Given as input a reduction rule r, two terminal graphs G_2 and G_1, a graph $G = G_2 \oplus H$, and a two-coloring of G, algorithm $A_{\mathcal{R}}$ can easily compute a two-coloring of $G_1 \oplus H$ using the given table: the algorithm looks which vertices of G_2 have which color, and looks up the corresponding coloring of G_1 in the table. Then it removes the inner vertices of G_2 from the solution and adds the inner vertices of G_1 in the correct way.

Algorithm $A_{\mathcal{I}}$ also uses a table: for the only element $H \in \mathcal{I}$, this table contains a two-coloring of H. See part II of Figure 4. Hence $(\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$ is a constructive reduction system for P defined by (D,Q). Note that both algorithms can be made to run in $O(1)$ time if we use a convenient data structure.

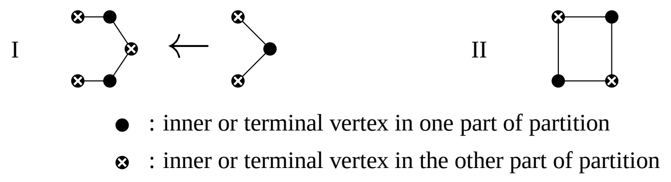
I
II
● : inner or terminal vertex in one part of partition
⊗ : inner or terminal vertex in the other part of partition

Figure 4: Example of tables used by $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ for constructive reduction system for two-colorability on cycles.

In order to make an efficient constructive reduction algorithm based on a constructive reduction system $(\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$, we want that algorithms $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ work efficiently. This is required in a special constructive reduction system.

Definition 3.2 (Special Constructive Reduction System). *Let P be a construction property defined by (D,Q). A constructive reduction system $(\mathcal{R},\mathcal{I},A_{\mathcal{R}},A_{\mathcal{I}})$ for P is a* special constructive reduction system *for P if*

1. *$(\mathcal{R},\mathcal{I})$ is a special reduction system for P (Definition 2.7), and*
2. *algorithms $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ run in $O(1)$ time.*

Note that the constructive reduction system we gave for two-colorability of cycles is a special constructive reduction system, since algorithms $A_{\mathcal{I}}$ and $A_{\mathcal{R}}$ as described take constant time, and we have shown that the reduction system depicted in Figure 3 is a special reduction system for the problem.

One way to obtain an algorithm $A_{\mathcal{R}}$ in a constructive reduction system which runs in $O(1)$ time is to ensure that $A_{\mathcal{R}}$ only has to change a solution locally, i.e. that the solution to construct only differs from the input solution in the part of the graph that was involved in the reduction. We use this technique in most of our algorithms.

Let P be a construction property defined by (D,Q) and let $(\mathcal{R},\mathcal{I},A_{\mathcal{R}},A_{\mathcal{I}})$ be a special constructive reduction system for P. The following algorithm computes for a given graph G a solution of G if one exists.

```
Algorithm Reduce-Construct($G$)
Input: Connected graph $G$
Output: $S \in D(G)$ for which $Q(G,S)$ holds if $P(G)$ holds, false otherwise
1.  (* Part 1 *)
2.  Apply as many reductions as possible on $G$ in the way of algorithm Reduce. Store the
    resulting sequence $(G_1^1,G_2^1),(G_1^2,G_2^2),\ldots,(G_1^t,G_i^t)$, where $t$ denotes the number of reduc-
    tions, and for each $i$, $1 \leq i \leq t$, in the $i$th reduction, $G_1^i$ is replaced by $G_2^i$. Let $G$ be the
    reduced graph.
3.  (* Part 2 *)
4.  if $G \notin \mathcal{I}$ then return false
5.  (* Construct initial solution *)
6.  $S \leftarrow A_{\mathcal{I}}(G)$
7.  for $i \leftarrow t$ downto 1
8.      do let $r = (H_1,H_2) \in \mathcal{R}$ such that $H_1$ and $G_1^i$ are isomorphic and $H_2$ and $G_2^i$ are iso-
           morphic.
9.         (* reconstruct solution *)
10.        $S \leftarrow A_{\mathcal{R}}(r,G_1^i,G_2^i,G,S)$
11.        (* undo $i$th reduction *)
12.        replace $G_2^i$ by $G_1^i$ in $G$
13. return $S$
```

It is clear from Lemma 2.2 and the definition of a constructive reduction system that algorithm Reduce-Construct is correct. Consider the running time of the algorithm. Part 1 takes $O(n)$ time, by Lemma 2.3. In part 2, the initial solution can be constructed in constant time, since algorithm $A_{\mathcal{I}}$ takes $O(1)$ time. Every undo-action also takes constant time: undoing a reduction can be done in the same way as applying it, which takes $O(1)$ time, and algorithm

A_R uses $O(1)$ time. Hence the complete algorithm takes $O(n)$ time. This proves the following theorem.

Theorem 3.1. *Let P be a construction property defined by the pair (D,Q). If we have a special constructive reduction system for P, then we have an algorithm which, given a connected graph G, returns a solution $S \in D(G)$ for which $Q(G,S)$ holds, if $P(G)$ holds, and* false *otherwise. The algorithm runs in $O(n)$ time and uses $O(n)$ space.*

3.2 Construction Problems for Graphs of Bounded Treewidth

In this section we show that algorithm Reduce-Construct can be used for a large class of construction properties on graphs of bounded treewidth.

For reasons of clarity, we only consider solution domains of the following form: there is a $t \geq 1$, such that for all graphs G, all elements of $D(G)$ are t-tuples $(S_1, S_2, \ldots, S_t)$, where for each i, $1 \leq i \leq t$, S_i is an element of $V(G)$, of $E(G)$, of $\mathcal{P}(V(G))$ or of $\mathcal{P}(E(G))$. If D is of this form, we say that D is a *t-vertex-edge-tuple* or, if t is not important, a *vertex-edge-tuple*. An example of a domain which is a t-vertex-edge-tuple is the domain D for which for each graph G, $D(G)$ contains all ordered t-partitions of $V(G)$, i.e. for each $S \in D(G)$, $S = (V_1, \ldots, V_t)$, where $V_1, \ldots, V_t$ partition $V(G)$.

If D is a t-vertex-edge-tuple with $t = 1$, each solution is of the form (S). In this case we usually omit the $()$.

Let D be some solution domain which is a t-vertex-edge-tuple. Let G and H be l-terminal graphs and let $S \in D(G \oplus H)$. We want to be able to restrict S to the terminal graphs G and H. For these restrictions, we use the notation $S[G]$ and $S[H]$, defined as follows. Suppose $S = (S_1, \ldots, S_t) \in D(G \oplus H)$. Then $S[G] = (S_1[G], \ldots, S_t[G])$, where for each i, $S_i[G]$ is defined as follows.

$$S_i[G] = \begin{cases} S_i \cap V(G) & \text{if domain of } S_i \text{ is } \mathcal{P}(V(G \oplus H)) \\ S_i \cap E(G) & \text{if domain of } S_i \text{ is } \mathcal{P}(E(G \oplus H)) \\ S_i & \text{if domain of } S_i \text{ is } V(G \oplus H) \text{ and } S_i \in V(G) \\ S_i & \text{if domain of } S_i \text{ is } E(G \oplus H) \text{ and } S_i \in E(G) \\ \varepsilon & \text{if domain of } S_i \text{ is } V(G \oplus H) \text{ and } S_i \notin V(G) \\ \varepsilon & \text{if domain of } S_i \text{ is } E(G \oplus H) \text{ and } S_i \notin E(G) \end{cases}$$

Note that with this definition, $S[G]$ does not contain any vertices or edges which are not in G.

Definition 3.3. *Let D be a vertex-edge-tuple. For each $l \geq 0$, and each l-terminal graph G, define*

$$D_{[]}(G) = \{S[G] \mid S \in D(G \oplus H) \textit{ for some } l\textit{-terminal graph } H\}.$$

Each $S \in D_{[]}(G)$ is called a partial solution *of G, and $D_{[]}$ is called the partial solution domain for D.*

Let D be a vertex-edge-tuple. Note that, for each two l-terminal graphs G and H ($l \geq 0$) and $S_G \in D_{[]}(G)$ and $S_H \in D_{[]}(H)$, there is at most one $S \in D(G \oplus H)$ such that $S[G] = S_G$ and $S[H] = S_H$.

Definition 3.4. *Let D be a vertex-edge-tuple. Let G and H be l-terminal graphs, let $S_G \in D_{[]}(G)$ and $S_H \in D_{[]}(H)$. If there is an $S \in D(G \oplus H)$ such that $S[G] = S_G$ and $S[H] = S_H$, then (G,S_G) and (H,S_H) are called* $\oplus$-compatible*, and we write $S_G \oplus S_H = S$.*

The intuition behind $\oplus$-compatibility is the following: if (G,S_G) and (H,S_H) are $\oplus$-compatible, then S_G and S_H can be 'glued' upon each other in order to get a solution in $G \oplus H$.

Let P be a construction property defined by (D,Q). Let G and H be terminals graphs, and let $S \in D_{[]}(G)$ and $S' \in D_{[]}(H)$. The value of $Q(G \oplus H, S \oplus S')$ is only defined if G and H are both l-terminal graphs for some $l \geq 0$, and (G,S) and (H,S') are $\oplus$-compatible. For shorter notation, we define $Q(G \oplus H, S \oplus S')$ to be false if G and H are not both l-terminal graphs for some $l \geq 0$, or if (G,S) and (H,S') are not $\oplus$-compatible.

Definition 3.5 (Compatibility). *Let D be a vertex-edge-tuple. Let G_1 and G_2 be l-terminal graphs for some $l \geq 0$, and let $S_1 \in D_{[]}(G_1)$ and $S_2 \in D_{[]}(G_2)$. (G_1,S_1) and (G_2,S_2) are* compatible *if for each l-terminal graph H and each $S \in D_{[]}(H)$, (G_1,S_1) is $\oplus$-compatible with (H,S) if and only if (G_2,S_2) is $\oplus$-compatible with (H,S).*

The intuition behind compatibility is the following: if (G_1,S_1) and (G_2,S_2) are compatible, then for each terminal graph H with the same number of terminals as G_1 (and G_2), any partial solution S_H in H that can be 'glued' upon S_1 can also be glued upon S_2, and vice versa.

Note that compatibility is an equivalence relation. The set of all equivalence classes of this relation is denoted by $\mathcal{C}_{\mathrm{cmp},l}$, for each l, and the equivalence classes are also called compatibility classes. Note that, for vertex-edge-tuples D, the equivalence relation is of finite index. For two equivalence classes C and C' of some equivalence relation which is a refinement of compatibility, we say that C and C' are $\oplus$-compatible if, for each $(G,S) \in C$ and $(H,S') \in C'$, (G,S) and (H,S') are $\oplus$-compatible.

Let P be a construction property defined by (D,Q), where D is a vertex-edge-tuple.

Definition 3.6. *For each $l \geq 0$, $\sim_{Q,l}$ is an equivalence relation on pairs of l-terminal graphs and partial solutions, which is defined as follows. Let G_1, G_2 be l-terminal graphs, and $S_1 \in D_{[]}(G_1)$ and $S_2 \in D_{[]}(G_2)$.*

$$\begin{array}{lll} (G_1,S_1) \sim_{Q,l} (G_2,S_2) & \Leftrightarrow & (G_1,S_1) \text{ and } (G_2,S_2) \text{ are compatible and} \\ & & \text{for all } l\text{-terminal graphs } H \text{ and all } S \in D_{[]}(H)\text{:} \\ & & \quad Q(G_1 \oplus H, S_1 \oplus S) \equiv Q(G_2 \oplus H, S_2 \oplus S) \end{array}$$

The set of equivalence classes of $\sim_{Q,l}$ is denoted by $\mathcal{C}_{Q,l}$, and for each l-terminal graph G and $S \in D_{[]}(G)$, the equivalence class of $\mathcal{C}_{Q,l}$ that contains (G,S) is denoted by $\mathrm{ec}_{Q,l}(G,S)$.

By $\sim_{rQ,l}$ we usually denote an equivalence relation which is a refinement of $\sim_{Q,l}$. By $\mathcal{C}_{rQ,l}$ we denote the set of equivalence classes of $\sim_{rQ,l}$, and for each l-terminal graph G and each $S \in D_{[]}(G)$, $\mathrm{ec}_{rQ,l}(G,S) = C$ if (G,S) is in equivalence class $C \in \mathcal{C}_{rQ,l}$.

Definition 3.7. *Let $\sim_{rQ,l}$ be a refinement of $\sim_{Q,l}$ for each $l \geq 0$. By $\approx_{rQ,l}$ we denote the equivalence relation on l-terminal graphs which is defined as follows. For every two l-terminal graphs G_1 and G_2,*

$$G_1 \approx_{rQ,l} G_2 \quad \Leftrightarrow \quad \{\, \mathrm{ec}_{rQ,l}(G_1,S_1) \mid S_1 \in D_{[]}(G_1)\,\} = \{\, \mathrm{ec}_{rQ,l}(G_2,S_2) \mid S_2 \in D_{[]}(G_2)\,\}.$$

Suppose P is a construction property defined by (D,Q). For each $k \geq 1$, let Q_k denote the property with for each graph G, each $S \in D(G)$, $Q_k(G,S)$ holds if and only if $Q(G,S) \wedge TW_k(G)$ holds. Note that P_k is the construction property defined by (D,Q_k).

For each $k \geq 1$, let $\sim_{rQk,l}$ be the refinement of $\sim_{rQ,l}$ which is defined as follows. For every two l-terminal graphs G_1 and G_2 and each $S_1 \in D_{[]}(G_1)$ and $S_2 \in D_{[]}(G_2)$,

$$(G_1,S_1) \sim_{rQk,l} (G_2,S_2) \quad \Leftrightarrow \quad (G_1,S_1) \sim_{rQ,l} (G_2,S_2) \wedge G_1 \sim_{TW_k,l} G_2.$$

Lemma 3.1. *Let $\sim_{rQ,l}$ be a refinement of $\sim_{Q,l}$, and let $k \geq 1$.*

1. *For each $l \geq 0$, $\approx_{rQ,l}$ is a refinement of $\approx_{Q,l}$.*
2. *For each $l \geq 0$, $\approx_{Q,l}$ is a refinement of $\sim_{P,l}$.*
3. *For each $l \geq 0$, if $\sim_{rQ,l}$ is finite, then $\approx_{rQ,l}$ is finite.*
4. *For each $l \geq 0$, if $\sim_{rQ,l}$ is finite, then $\approx_{rQk,l}$ is finite.*

Proof.
1. Follows directly from the definition of $\approx_{rQ,l}$.

2. Follows from the fact that for every two l-terminal graphs G_1 and G_2, if $G_1 \approx_{Q,l} G_2$, then for each $S_1 \in D_{[]}(G_1)$ there is an $S_2 \in D_{[]}(G_2)$ such that $(G_1,S_1) \sim_{Q,l} (G_2,S_2)$.

3. The number of equivalence classes of $\approx_{rQ,l}$ is at most $2^{|C_{rQ,l}|}$.

4. Follows from Lemmas 2.5 and 2.4. □

The next theorem is the analog of Theorem 2.2 for construction properties: we give a set of conditions for a construction property P, and we show that these conditions are sufficient for proving the existence of a special constructive reduction system for P_k for any $k \geq 1$.

Theorem 3.2. *Let P a construction property defined by (D,Q), suppose that D is a vertex-edge-tuple. If $\sim_{Q,l}$ has finitely many equivalence classes for each $l \geq 0$, then for each $k \geq 1$, there exists a special constructive reduction system $(\mathcal{R}, \mathcal{I}, \mathcal{A}_{\mathcal{R}}, \mathcal{A}_{\mathcal{I}})$ for P_k defined by (D,Q_k).*

If, in addition, Q and a finite refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ are effectively decidable, then such a special constructive reduction system can effectively be constructed. Moreover for each rule $r = (H_1,H_2)$ in this system, $H_1 \approx_{rQ,l} H_2$.

Proof. Let $k \geq 1$. Since $|C_{Q,l}|$ is finite, $\approx_{Qk,l}$ has a finite number of equivalence classes, and it is a refinement of $\sim_{P_k,l}$. Let $(\mathcal{R}, \mathcal{I})$ be a special reduction system for P_k, such that for each rule $(H_1,H_2) \in \mathcal{R}$, $H_1 \approx_{Qk,l} H_2$. By Corollary 2.1, such a system exists, and it can be effectively constructed if Q is effectively decidable and a finite refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ is effectively decidable, since in that case, P and $\approx_{rQk,l}$ are effectively decidable as well (for each constructed rule (H_1,H_2), $H_1 \approx_{rQk,l} H_2$).

We now describe algorithms $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ for which $(\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$ is a special constructive reduction system for P_k. Both algorithm $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ use a table (see also the example for two-colorability in Figure 4).

For algorithm $A_{\mathcal{R}}$, we make a table which contains for each rule $r = (H_1,H_2) \in \mathcal{R}$ and each $S_2 \in D_{[]}(H_2)$ an $S_1 \in D_{[]}(H_1)$ such that $(H_1,S_1) \sim_{Qk,l} (H_2,S_2)$. This table is computed as follows. For each reduction rule (H_1,H_2) in $\mathcal{R}$, we construct all $S_1 \in D_{[]}(H_1)$ and all $S_2 \in D_{[]}(H_2)$. Then, for each $S_2 \in D_{[]}(H_2)$, we pick one $S_1 \in D_{[]}(H_1)$ for which $(H_1,S_1) \sim_{rQ,l} (H_2,S_2)$. Note that these tables have $O(1)$ size, and can be effectively constructed if Q is effectively decidable and a finite refinement $\sim_{rQ,l}$ is effectively decidable.

For algorithm $A_{\mathcal{I}}$, we make a table which contains for each $H \in \mathcal{I}$ a solution S of H. This is done as follows. For each $H \in \mathcal{I}$, we construct all $S \in D(H)$, and we pick one such S for which $Q(H,S)$ holds. These tables can be effectively constructed if Q is effectively decidable.

In order to make algorithms $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ run in $O(1)$ time, we use a data structure for storing tuples $S = (S_1,\dots,S_t) \in D_{[]}(G)$ which consists of an array of t data structures, one for each S_i. If S_i is a set of vertices or edges, then these vertices or edges are put in a (doubly linked) list. If S_i is a vertex or edge, or ε, then this vertex or edge or ε is stored. Furthermore, we keep a pointer from each vertex and edge in the graph to each place in the data structure where this vertex or edge occurs. There are at most t of these pointers for each vertex and each edge. This implies that algorithm $A_{\mathcal{I}}$ can be made to run in $O(1)$ time.

Consider algorithm $A_{\mathcal{R}}$. Suppose we have a rule $r = (H_1,H_2)$ to undo, and we have terminal graphs G_1 and G_2 (isomorphic to H_1 and H_2, respectively), a graph $G = G_2 \oplus H$ for some H, and an $S \in D(G)$ for which $Q(G,S)$ holds. Now we compute an $S' \in D(G_1 \oplus H)$ such that $Q(G_1 \oplus H, S')$ holds, as follows. First, we compute $S[G_2]$ as follows. Make a new data structure for $S[G_2]$ with $S_i[G_2]$ empty for each i. For each vertex v in G_2, follow the pointers from v to the places in which it occurs in S, and check in which part S_i of S it occurs. Then add v to $S_i[G_2]$. Do the same for all edges. Then for each i, check if S_i is a set of vertex or edge, but there is no vertex or edge in the data structure at the location of $S_i[G_2]$, and if so, add ε to $S_i[G_2]$. This can all be done in constant time, since G_2 has constant size, and each vertex or edge occurs at most once in each S_i, so at most t times in S.

Next, find an $S' = (S'_1,\dots,S'_t) \in D_{[]}(G_1)$ with the table that is kept for rule r (note that $(G_1,S') \sim_{Qk,l} (G_2,S[G_2])$). This can again be done in constant time.

Then compute $S' \oplus S[H]$ as follows. Remove all vertices and edges of G_2 from S. Next, for each i, $1 \le i \le t$, append the list S'_i to the list $S_i[H]$ (do not copy ε). The resulting data structure represents $S' \oplus S[H]$. Hence algorithm $A_{\mathcal{R}}$ uses $O(1)$ time. □

As an important special case, we now consider the MS-definable construction properties. The construction properties defined by (D,Q), where D is a vertex-edge-tuple and Q is an MS-definable extended graph property, correspond exactly to the MS-definable construction problems (see e.g. [3]). These MS-definable construction problems can be solved in $O(n)$ time and space for graphs of bounded treewidth if a tree decomposition of bounded width the input graph is given.

Theorem 3.3. *Let P be a construction property defined by (D,Q), where D is a vertex-edge-*

tuple and Q is MS-definable. For each $k \geq 1$ there is a special constructive reduction system for P_k, which can be effectively constructed if a definition of Q in MSOL is known.

Proof. In [9] is was shown that for each $k \geq 1$, there is a homomorphism h, mapping each pair (G,S), where either G is an ordinary graph and $S \in D(G)$ or G is an l-terminal graph, $l \leq k$, and $S \in D_{[]}(G)$, to an element of a finite set A_k, such that the following conditions hold.

1. For every two graphs G_1 and G_2, and each $S_1 \in D(G_1)$ and $S_2 \in D(G_2)$, if $h(G_1,S_1) = h(G_2,S_2)$, then $Q(G_1,S_1) = Q(G_2,S_2)$.
2. There is a function $f_\oplus : A_k \times A_k \to A_k$, such that for each $l \leq k$, every two l-terminal graphs G and H, and each $S \in D_{[]}(G)$ and $S' \in D_{[]}(H)$, if (G,S) and (H,S') are $\oplus$-compatible, then
$$h(G \oplus H, S \oplus S') = f_\oplus(h(G,S),h(H,S')).$$

This homomorphism can be computed from an MSOL predicate for Q.

For each $l \geq 0$, each l-terminal graph G and $S \in D_{[]}(G)$, let $\mathrm{ec}_l(G,S) = (h(G,S),C)$, where $C \in C_{\mathrm{cmp},l}$ is such that (G,S) belongs to compatibility class C. Furthermore, let $C_l = A_k \times C_{\mathrm{cmp},l}$, and let $(G_1,S_1) \sim_l (G_2,S_2)$ if and only if $\mathrm{ec}_l(G_1,S_1) = \mathrm{ec}_l(G_2,S_2)$. Since $|A_k|$ and $|C_{\mathrm{cmp},l}|$ are both finite, $|C_l|$ is also finite. We now show that $\sim_l$ is a refinement of $\sim_{Q,l}$.

Let $l \geq 0$, let G_1 and G_2 be l-terminal graphs, and let $S_1 \in D_{[]}(G_1)$ and $S_2 \in D_{[]}(G_2)$, such that $(G_1,S_1) \sim_l (G_2,S_2)$. We have to show that for all l-terminal graphs H and all $S \in D_{[]}(H)$, $Q(G_1 \oplus H, S_1 \oplus S) = Q(G_2 \oplus H, S_2 \oplus S)$. Let H be an l-terminal graph, and let $S \in D_{[]}(H)$ such that (G_1,S_1) and (H,S) are $\oplus$-compatible. Then, since $h(G_1,S_1) = h(G_2,S_2)$,
$$\begin{aligned} h(G_1 \oplus H, S_1 \oplus S) &= f_\oplus(h(G_1,S_1),h(H,S)) \\ &= f_\oplus(h(G_2,S_2),h(H,S)) \\ &= h(G_2 \oplus H, S_2 \oplus S). \end{aligned}$$

Hence $Q(G_1 \oplus H, S_1 \oplus S) = Q(G_2 \oplus H, S_2 \oplus S)$. This shows that the conditions of Theorem 3.2 hold. □

Theorem 3.3 implies that for each MS-definable construction property, there is a linear time and space algorithm which solves P constructively on graphs of bounded treewidth, without making use of a tree decomposition of the input graph. For instance, this gives linear time algorithms for the constructive versions of HAMILTONIAN CIRCUIT and k-COLORABILITY for fixed k, all on graphs of bounded treewidth.

4 Reduction Algorithms for Optimization Problems

In this section we show how the idea of reduction algorithms can be extended to optimization problems. The general idea is to extend the reduction algorithm as follows. During the reductions, an integer is kept which is initially zero. Each time a reduction is applied, this integer is increased (or possibly decreased) with some specified amount. When no more reductions are possible, the integer represents the optimal value of the problem.

In Section 4.1 we show how this algorithm can be made to work: we extend reduction systems to reduction-counter systems and give an efficient reduction algorithm based on such a system. In Section 4.2 we show that this algorithm can be used for a large class of optimization problems on graphs of small treewidth.

4.1 Reduction-Counter Systems and Algorithms

Let Φ be a function which maps each graph to a value in $\mathbb{Z} \cup \{\mathsf{false}\}$ (we assume that isomorphic graphs are mapped to the same value). Typically, Φ will be an optimization problem like MAX INDEPENDENT SET. We will call Φ a *graph optimization problem*. The value false is used to denote that a certain condition does not hold, i.e. that there is no optimum for a graph. Let $\mathcal{Z}$ denote the set $\mathbb{Z} \cup \{\mathsf{false}\}$. Define addition on $\mathcal{Z}$ as follows: if $i, j \in \mathbb{Z}$, then we take for $i + j$ the usual sum, and for all $i \in \mathcal{Z}$, $i + \mathsf{false} = \mathsf{false} + i = \mathsf{false}$.

Instead of reduction rules, we use *reduction-counter* rules for graph optimization problems.

Definition 4.1 (Reduction-Counter Rule). *A* reduction-counter rule *is a pair (r, i), where r is a reduction rule and i an integer.*

A match to a reduction-counter rule (r, i) in a graph G is a match to r in G.

If G contains a match to a reduction-counter rule $r' = (r, i)$, then an application of r' to a graph G and an integer counter cnt is an operation which applies r to G and replaces cnt by $cnt + i$. An application of a reduction-counter rule is also called a reduction.

Let G and G' be two graphs. If there is a reduction-counter rule r such that applying r to G and some counter cnt can result in G', then we write $G \xrightarrow{r'} G'$. If we have a set $\mathcal{R}$ of reduction-counter rules, we write $G \xrightarrow{\mathcal{R}} G'$ if there exists an $r \in \mathcal{R}$ with $G \xrightarrow{r} G'$. If a graph G has no match in $\mathcal{R}$, then we say that G is irreducible (w.r.t. $\mathcal{R}$).

We extend the notions of safeness, completeness and decrease to reduction-counter rules.

Definition 4.2. *Let Φ be a graph optimization problem and $\mathcal{R}$ a set of reduction-counter rules.*

- *$\mathcal{R}$ is* safe *for Φ if, whenever $G \xrightarrow{r} G'$ for some $r = (r', i) \in \mathcal{R}$, then $\Phi(G) = \Phi(G') + i$.*
- *$\mathcal{R}$ is* complete *for Φ if the set $\mathcal{I}$ of irreducible graphs G for which $\Phi(G) \neq \mathsf{false}$ is finite.*
- *$\mathcal{R}$ is* decreasing *if whenever $G \xrightarrow{\mathcal{R}} G'$, then G' contains fewer vertices than G.*

Definition 4.3 (Reduction-Counter System). *A* reduction-counter system *for a graph optimization problem Φ is a triple $(\mathcal{R}, \mathcal{I}, \phi)$, where $\mathcal{R}$ is a finite set of reduction-counter rules which is safe, complete and decreasing for Φ, $\mathcal{I}$ is the set of graphs G which are irreducible and for which $\Phi(G) \neq \mathsf{false}$, and ϕ is a function mapping each graph $G \in \mathcal{I}$ to the value $\Phi(G)$.*

As a simple example we give a reduction-counter system for the optimization problem MAX INDEPENDENT SET on cycles: for each graph G, if G is a cycle then $\Phi(G)$ is the size of a maximum independent set in G, otherwise $\Phi(G) = \mathsf{false}$. Let $\mathcal{R} = \{(r, 1)\}$, where r is the reduction rule depicted in Figure 5, let $\mathcal{I} = \{C_3, C_4\}$, where C_3 and C_4 are the cycles on three and four vertices (see Figure 5), and let $\phi(C_3) = 1$, $\phi(C_4) = 2$. It can easily be seen that $(\mathcal{R}, \mathcal{I}, \phi)$ is a reduction-counter system for Φ.

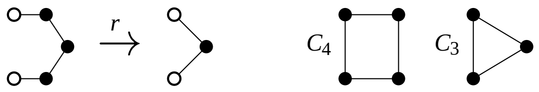

Figure 5: A reduction rule and a set of irreducible graphs that form the basis for a reduction-counter system for MAX INDEPENDENT SET on cycles.

Let Φ be a graph optimization problem. Let P be the graph property with for each graph G, $P(G) = \mathsf{true}$ if $\Phi(G) \in \mathbb{Z}$, and $P(G) = \mathsf{false}$ if $\Phi(G) = \mathsf{false}$. We call P the *derived* graph property (of Φ). From a reduction-counter system $(\mathcal{R}, \mathcal{I}, \phi)$ for Φ, we can derive a reduction system for P: let $\mathcal{R}' = \{r \mid (r,i) \in \mathcal{R}$ for some $i \in \mathbb{Z}\}$. Then $(\mathcal{R}', \mathcal{I})$ is a reduction system for P. We call this system the *derived reduction system* (from $(\mathcal{R}, \mathcal{I}, \phi)$).

If we are given a reduction-counter system $\mathcal{S} = (\mathcal{R}, \mathcal{I}, \phi)$ for a graph optimization problem Φ, we can again use a reduction algorithm to solve Φ in polynomial time. Let $\mathcal{S}'$ denote the derived reduction system. A reduction algorithm based on $\mathcal{S}$ is a modification of a reduction algorithm for the derived graph property based on $\mathcal{S}'$: instead of repeatedly applying reduction rules from $\mathcal{S}'$ on the input graph G, repeatedly apply reduction-counter rules from $\mathcal{S}$ on the graph G and a counter *cnt*. Initially, *cnt* is set to zero.

Let G_j denote the graph after the jth reduction is done, and let cnt_j denote the value of the counter at this moment (hence G_0 denotes the input graph, and $cnt_0 = 0$). It is important to note that the sum $\Phi(G_j) + cnt_j$ is invariant during the reduction process, because of the safeness property. Thus, at each moment in the reduction algorithm, $\Phi(G_0) = \Phi(G_j) + cnt_j$. Hence, when the reduction process stops after t iterations, because G_t is irreducible, then $\Phi(G_0) \in \mathbb{Z}$ if and only if $G_t \in \mathcal{I}$ (or, more precisely, G is isomorphic to a graph $H \in \mathcal{I}$). Hence if $G_t \in \mathcal{I}$, then $\Phi(G_0) = \phi(G_t) + cnt_t$, otherwise, $\Phi(G_0) = \mathsf{false}$.

Definition 4.4 (Special Reduction-Counter System). *A special reduction-counter system is a reduction-counter system of which the derived reduction system is special (Definition 2.7).*

Note that the reduction-counter system for MAX INDEPENDENT SET on cycles that we have given above is also a special reduction-counter system for this problem.

Clearly, if we have a special reduction-counter system for a graph optimization problem Φ, then we can apply the modifications described above to algorithm Reduce in order to get a linear time algorithm for solving Φ on connected graphs.

Theorem 4.1. *Let Φ be a graph optimization problem. If we have a special reduction-counter system for Φ, then we have an algorithm which, for each connected graph G, computes $\Phi(G)$ in $O(n)$ time with $O(n)$ space.*

4.2 Optimization Problems for Graphs of Bounded Treewidth

In this section, we derive a similar result as Theorem 2.2 for reduction-counter systems.

In analogy to $\sim_{P,l}$ for graph properties P, we define an equivalence relation $\sim_{\Phi,l}$ for graph optimization problems Φ.

Definition 4.5. *For a graph optimization problem Φ the equivalence relation $\sim_{\Phi,l}$ on l-terminal graphs is defined as follows. Let G_1 and G_2 be two l-terminal graphs.*

$$G_1 \sim_{\Phi,l} G_2 \quad \Leftrightarrow \quad \text{there is an } i \in \mathbb{Z} \text{ such that for all } l\text{-terminals graphs } H: \\ \Phi(G_1 \oplus H) = \Phi(G_2 \oplus H) + i.$$

Optimization problem Φ is of finite integer index *if $\sim_{\Phi,l}$ is finite for each fixed l.*

Note that a if reduction-counter rule $((H,H'),i)$ is safe for a graph optimization problem Φ, then $H \sim_{\Phi,l} H'$. Furthermore, if $H \sim_{\Phi,l} H'$ for two l-terminal graphs H and H', then there is an $i \in \mathbb{Z}$ for which the reduction-counter rule $((H,H'),i)$ is safe for Φ. Note furthermore that, for each $l \geq 0$, $\sim_{\Phi,l}$ is a refinement of $\sim_{P,l}$, where P is the derived graph property of Φ. Hence if Φ is of finite integer index, then the derived property P is of finite index.

For any graph optimization problem Φ and any integer $k \geq 1$, Φ_k is the graph optimization problem with for each graph G,

$$\Phi_k(G) = \begin{cases} \text{false} & \text{if } \operatorname{tw}(G) > k \\ \Phi(G) & \text{otherwise.} \end{cases}$$

From Lemma 2.4 and Lemma 2.5 it follows that, if Φ is of finite integer index, then for each $k \geq 1$, Φ_k is of finite integer index.

The following theorem is the analog of Theorem 2.2 for finite integer index problems.

Theorem 4.2. *Let Φ is a graph optimization problem of finite integer index. For each integer $k \geq 1$ there exists a special reduction-counter system for Φ_k.*

If Φ is also effectively computable and there is an equivalence relation $\sim_l$, for each $l \geq 0$, which is a finite refinement of $\sim_{\Phi,l}$ and is effectively decidable, then such a special reduction-counter system $\mathcal{S}$ can effectively be constructed. Moreover, for each reduction-counter rule $((H,H'),i)$ in $\mathcal{S}$, $H \sim_l H'$.

Proof. Let $k \geq 1$. Let P be the derived graph property of Φ. Since for each $l \geq 0$, $\sim_{\Phi_k,l}$ is a refinement of $\sim_{P_k,l}$, Corollary 2.1 implies that there is a special reduction system $\mathcal{S} = (\mathcal{R}, \mathcal{I})$ for P, such that for each $(H,H') \in \mathcal{R}$, $H \sim_{\Phi_k,l} H'$. We show that we can construct a special reduction-counter system for Φ for which $\mathcal{S}$ is the derived reduction system. For each reduction rule (H,H'), make a reduction-counter rule $((H,H'),i)$, where $i = 0$ if for all G, $\Phi(H \oplus G) = \text{false}$ (and hence $\Phi(H' \oplus G) = \text{false}$), and $i = \Phi(H \oplus G) \Leftrightarrow \Phi(H' \oplus G)$ for some G such that $\Phi(H \oplus G) \in \mathbb{Z}$ otherwise. Let $\mathcal{R}'$ denote the set of all these reduction-counter rules. Let $\phi : \mathcal{I} \to \mathbb{Z}$ be the function mapping each graph $G \in \mathcal{I}$ to its value $\Phi(G)$. Then $(\mathcal{R}', \mathcal{I}, \phi)$ is a special reduction-counter system for Φ.

If Φ is effectively computable and we have a refinement $\sim_l$ of $\sim_{\Phi,l}$, for each $l \geq 0$, then Φ_k is effectively computable and P and P_k are effectively decidable. Hence we can effectively construct a special reduction system $(\mathcal{R}, \mathcal{I})$ for P_k, such that for each rule (H,H'), $H \sim_l H'$. Furthermore, we can turn this reduction system in a special reduction-counter system $(\mathcal{R}', \mathcal{I}, \phi)$ for Φ in the following way. The function ϕ can be computed by simply computing $\Phi(G)$ for each $G \in \mathcal{I}$.

For each reduction rule $r = (H,H') \in \mathcal{R}$, we compute an integer i such that (r,i) is a safe reduction-counter rule in $\mathcal{R}$. Suppose H and H' are l-terminal graphs. Let $\mathcal{G}$ be a finite class of l-terminal graphs containing at least one terminal graph of each equivalence class of $\sim_{\Phi,l}$. Such a set $\mathcal{G}$ can be effectively computed, similar as for finite index problems (use techniques similar as in [2, 16].) Now if there is a $G \in \mathcal{G}$ for which $\Phi(H \oplus G) \in \mathbb{Z}$, then take any such G and let $i = \Phi(H \oplus G) \Leftrightarrow \Phi(H' \oplus G)$. Note that, since $H \sim_{\Phi,l} H'$, for each $G \in \mathcal{G}$ with $\Phi(G \oplus H) \in \mathbb{Z}$, $\Phi(G \oplus H) \Leftrightarrow \Phi(G \oplus H')$ has the same value, hence this gives a proper value. If $\mathcal{G}$ contains no graph G for which $\Phi(H \oplus G) \in \mathbb{Z}$, then let $i = 0$. Note that in this case, for every l-terminal graph G, $\Phi(H \oplus G) = \Phi(H' \oplus G) = \mathsf{false}$, and hence $\Phi(H \oplus G) = \mathsf{false} = \mathsf{false} + 0 = \Phi(H' \oplus G) + i$. Let $\mathcal{R}'$ be the set of all reduction-counter rules that are found this way. $\square$

Unfortunately, we can not apply Theorem 4.2 to all MS-definable graph optimization problems (see e.g. [3] for a definition). Hence the analog of Corollary 2.2 does not hold for optimization problems. However, there are a number of problems for which we can prove that they are of finite integer index. We give them in the next theorem. In Section 5.2 we prove that these problems are of finite integer index (Theorem 5.3). These proofs make use of techniques introduced for constructive optimization problems in Section 5. Definitions of the problems can also be found in Theorem 5.3.

Theorem 4.3. *The following problems are of finite integer index:* MAX INDUCED d-DEGREE SUBGRAPH *for all* $d \geq 1$, MAX INDEPENDENT SET, MIN VERTEX COVER, MIN p-DOMINATING SET *for all* $p \geq 1$, MAX CUT *on graphs with bounded degree,* MIN PARTITION INTO CLIQUES, MIN HAMILTONIAN PATH COMPLETION, *and* MAX LEAF SPANNING TREE.

As said before, there are a number of optimization problems which are not of finite integer index, although the problems are MS-definable, and thus standard methods can be used to solve these problems in $O(n)$ time sequentially if a tree decomposition of the input graph is given. We state a number of these problems in the next theorem. We prove it only for one problem; the other proofs are similar, and can be found in [11].

Theorem 4.4. *The following problems are not of finite integer index.*

MAX CUT: *given a graph G, find a partition (V_1,V_2) of $V(G)$ such that the number of edges with one end point in V_1 and one in V_2 is maximum.*

MIN COVERING BY CLIQUES: *given a graph G, find a set of cliques in G of minimum cardinality, such that each edge of G is contained in at least one clique.*

LONGEST PATH: *given a graph G, find a path in G of maximum length.*

LONGEST CYCLE: *given a graph G, find a cycle in G of maximum length.*

Proof. We only give the proof of MIN COVERING BY CLIQUES. For each graph G, let $\Phi(G)$ denote the minimum number of cliques to cover G. We show that $\sim_{\Phi,l}$ has infinitely many equivalence classes for some $l \geq 0$ by giving an infinite class of graphs and showing that the elements of this class are pairwise not equivalent.

For each $n \geq 1$, let G_n be the two-terminal graph with (see also Figure 6)

$$V(G_n) = \{x_1, x_2\} \cup \{a_1, \ldots, a_n\}, \text{ and}$$
$$E(G_n) = \{\{x_i, a_j\} \mid 1 \leq i \leq 2 \wedge 1 \leq j \leq n\}.$$

Vertices x_1 and x_2 are the first and the second terminal, respectively.

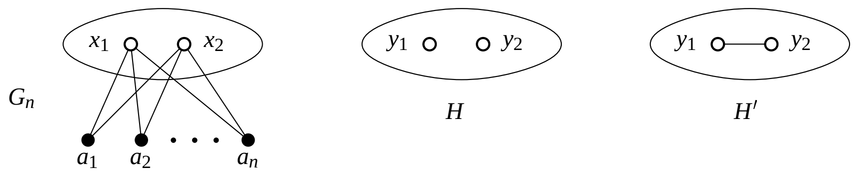

Figure 6: The graphs G_n ($n \geq 2$), H and H' for MIN COVERING BY CLIQUES.

Let $G = \{G_n \mid n \geq 1\}$. We show that for each $G_n, G_m \in G$, if $n \neq m$, then $G_n \not\sim_{\Phi,2} G_m$.

Let H be the two-terminal graph consisting of terminals y_1 and y_2 and no edges, and let H' be the two-terminal graph consisting of terminals y_1 and y_2 and edge $\{y_1, y_2\}$ (see Figure 6).

For each i, $i \geq 1$, $\Phi(G_i \oplus H) = |E(G_i)| = 2i$, since $G_i \oplus H$ contains no cliques of more than two vertices. Furthermore, $\Phi(G_i \oplus H') = |\{\{x_1, x_2, a_j\} \mid 1 \leq j \leq n\}| = i$. This means that for all n and m, $n \neq m$,

$$\Phi(G_n \oplus H) \Leftrightarrow \Phi(G_m \oplus H) = 2n \Leftrightarrow 2m \neq n \Leftrightarrow m = \Phi(G_n \oplus H') \Leftrightarrow \Phi(G_m \oplus H'),$$

and hence $G_n \not\sim_{\Phi,l} G_m$. This shows that the number of equivalence classes of $\sim_{\Phi,l}$ is infinite for some l. $\square$

5 Reduction Algorithms for Constructive Optimization Problems

In this section we show how the idea of constructive reduction algorithms and of reduction algorithms for optimization problems can be combined for constructive optimization problems. We start with a definition of a constructive reduction-counter system and an efficient reduction algorithm for constructive optimization problems. After that, we show that this algorithm can be used to solve a large class of constructive optimization problems on graphs of bounded treewidth.

5.1 Constructive Reduction-Counter Systems and Algorithms

Many graph optimization problems are of the form

$$\Phi(G) = \text{opt}\{z(S) \mid S \in D(G) \wedge Q(G, S)\},$$

where D is a solution domain, for each $S \in D(G)$, z is a function from $D(G)$ to $\mathbb{Z}$, and either opt = max or opt = min. (If there is no $S \in D(G)$ for which $Q(G, S)$ holds, then we define $\Phi(G)$ to be false.) If Φ is of this form, then we say Φ is a constructive optimization problem defined by the quadruple (D,Q,z,opt). MAX INDEPENDENT SET is an example of such an optimization problem: for this problem, we can choose opt $=$ max, $D(G) = \mathcal{P}(V(G))$, $Q(G,S)$ holds if and only if for each $v,w \in S$, $\{v,w\} \notin E(G)$, and $z(S) = |S|$.

In this section, we consider reduction algorithms for constructive optimization problems Φ which return the value of $\Phi(G)$ for an input graph G, and also construct an optimal solution for G, i.e. a solution $S \in D(G)$ for which $Q(G,S)$ holds and $z(S) = \Phi(G)$ (if $\Phi(G) \neq \mathsf{false}$). We again only consider solution domains D which are vertex-edge-tuples. We first define the constructive version of a reduction-counter system.

Definition 5.1 (Constructive Reduction-Counter System). *Let Φ be a constructive optimization problem defined by (D,Q,z,opt). A* constructive reduction-counter system *for Φ is a quadruple $(\mathcal{R},\mathcal{I},\phi,A_{\mathcal{R}},A_{\mathcal{I}})$, where*

- *$(\mathcal{R},\mathcal{I},\phi)$ is a reduction-counter system for Φ (Definition 4.3),*
- *$A_{\mathcal{R}}$ is an algorithm which, given*
  - *a reduction rule $r = (H_1,H_2) \in \mathcal{R}$,*
  - *two terminal graphs G_1 and G_2, such that G_1 is a isomorphic to H_1 and G_2 is isomorphic to H_2,*
  - *a graph G with $G = G_2 \oplus H$ for some H, and*
  - *an $S \in D(G)$ for which $Q(G,S)$ holds and $z(S) = \Phi(G)$,*

  computes an $S' \in G_1 \oplus H$ for which $Q(G_1 \oplus H, S')$ holds and $z(S') = \Phi(G_1 \oplus H)$,
- *$A_{\mathcal{I}}$ is an algorithm which, given a graph G which is isomorphic to some $H \in \mathcal{I}$, computes an $S \in D(G)$ for which $Q(G,S)$ holds and $z(S) = \Phi(G)$.*

As an example, consider the optimization problem Φ defined as follows. For each graph G, $\Phi(G)$ is the maximum size of an independent set if G is a cycle, $\Phi(G) = \mathsf{false}$ otherwise (see Section 4.1). Consider the constructive version of Φ defined by $(D,Q,z,\max)$, where D, Q and z are defined as follows. For each graph G, $D(G) = \mathcal{P}(V(G))$, and for each $S \in D(G)$, $Q(G,S)$ holds if and only if G is a cycle and S is an independent set of G, and $z(S) = |S|$.

We extend the reduction-counter system for Φ depicted in Figure 5 to a constructive reduction-counter system for Φ. Therefore, we again use the table method used in the proof of Theorem 3.2. For algorithm $A_{\mathcal{R}}$, we make a table which contains the following information. For the only reduction rule $r = (H_1,H_2) \in \mathcal{R}$ and each independent set S_2 of H_2 for which there is a maximum independent set S in some graph $H_2 \oplus H$ with $S_2 = S \cap V(H_2)$, the table contains an independent set S_1 of H_1 such that S_1 and S_2 contain the same terminals and $|S_1| = |S_2| + 1$. All these cases are depicted in part I of Figure 7 (symmetric cases are given only once). Note that algorithm $A_{\mathcal{R}}$ can be made to run in $O(1)$ time with this table, since it only has to remove inner vertices of H_2 from the independent set of the old graph and add some inner vertices of H_1 to the independent set of the new graph.

For algorithm $A_{\mathcal{I}}$, we make a table which contains for each $H \in \mathcal{I}$ a maximum independent set of H (see part II of Figure 7). Algorithm $A_{\mathcal{I}}$ also uses $O(1)$ time. It can be seen that $(\mathcal{R},\mathcal{I},\phi,A_{\mathcal{R}},A_{\mathcal{I}})$ is a constructive reduction-counter system for Φ defined by $(D,Q,z,\max)$.

Let Φ be a constructive optimization problem defined by (D,Q,z,opt). Let P be the construction property defined by (D,Q). We call P the *derived construction property*. From a

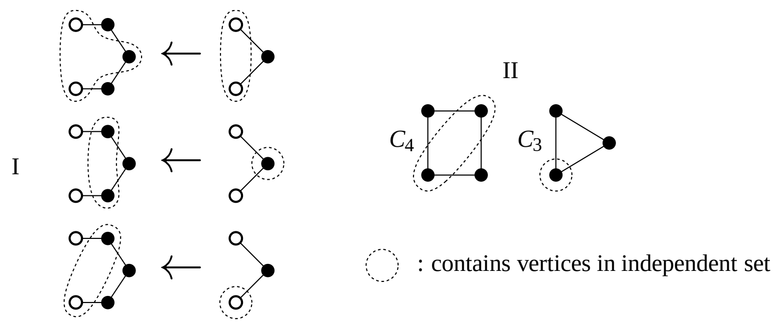

Figure 7: Tables for algorithms A_R and A_I for constructive reduction-counter system for MAX INDEPENDENT SET on cycles.

constructive reduction-counter system $(\mathcal{R}, \mathcal{I}, \phi, A_{\mathcal{R}}, A_{\mathcal{I}})$ for Φ we can derive a constructive reduction system $\mathcal{S}$ for P: let $\mathcal{R}' = \{r \mid (r,i) \in \mathcal{R}\}$, and let $\mathcal{S} = (\mathcal{R}', \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$. We call $\mathcal{S}$ the *derived constructive reduction system*.

Definition 5.2 (Special Constructive Reduction-Counter System). *A special constructive reduction-counter system is a constructive reduction-counter system whose derived constructive reduction system is special.*

Note that the constructive reduction-counter system that we gave for MAX INDEPENDENT SET on cycles is special.

Let Φ be a constructive optimization problem defined by (D, Q, z, opt), such that D is a vertex-edge-tuple. Let $\mathcal{S} = (\mathcal{R}, \mathcal{I}, \phi, A_{\mathcal{R}}, A_{\mathcal{I}})$ be a special constructive reduction-counter system for Φ. We can modify algorithm Reduce-Construct (Section 3.1) to obtain a constructive reduction algorithm for Φ based on $\mathcal{S}$: in part 1, use the reduction-counter algorithm as described in Section 4.1 instead of algorithm Reduce. In Part 2, line 6 of algorithm Reduce-Construct, store the value $\phi(G)$ in some variable *opt*. In line 13, return with S the value *opt*.

Hence we have the following result.

Theorem 5.1. *Let Φ be a constructive optimization problem defined by (D, Q, z, opt). If we have a special constructive reduction-counter system for Φ then we have an algorithm which, given any connected graph G, computes $\Phi(G)$ and, if $\Phi(G) \neq \mathsf{false}$, computes an $S \in D(G)$ such that $Q(G,S)$ holds and $z(S) = \Phi(G)$. The algorithm uses $O(n)$ time and space.*

5.2 Constructive Optimization Problems for Graphs of Bounded Treewidth

In this section we give a number of conditions that are sufficient for constructive optimization problems on graphs of bounded treewidth in order to assure that there is a special constructive reduction system. We also show that these conditions hold for a number of problems.

Let (D, Q, z, opt) define a constructive optimization problem Φ and suppose D is a vertex-edge-tuple. For each $l \geq 0$, let $\sim_{rQ,l}$ be a refinement of $\sim_{Q,l}$.

Let G be a terminal graph. We want to be able to compare the quality of two partial solutions S and S' for which $(G,S) \sim_{rQ,l} (G,S')$. More formally, we want that there is an integer

$i \in \mathbb{Z}$ such that for each terminal graph H and each $S_H \in D_{[]}(H)$ for which $Q(G \oplus H, S \oplus S_H)$ holds, $z(S \oplus S_H) \Leftrightarrow z(S' \oplus S_H) = i$. Therefore, we define an extension of the function z to the domain of terminal graphs.

Definition 5.3. *Let $\overline{z}$ be a function which, for each terminal graph G and each $S \in D_{[]}(G)$, maps S to a value in $\mathbb{Z}$. Function $\overline{z}$ is an* extension *of z with respect to $\{\sim_{rQ,l} \mid l \geq 0\}$ if, for each $l \geq 0$, and each $C, C' \in \mathcal{C}_{rQ,l}$ for which C and C' are $\oplus$-compatible, there is a constant $d_l(C,C') \in \mathbb{Z}$ such that the following holds. For every two l-terminal graphs G and H and all $S_G \in D_{[]}(G)$ and $S_H \in D_{[]}(H)$ such that $\mathrm{ec}_{rQ,l}(G,S_G) = C$ and $\mathrm{ec}_{rQ,l}(H,S_H) = C'$,*

$$Q(G \oplus H, S_G \oplus S_H) \quad \Rightarrow \quad z(S_G \oplus S_H) = \overline{z}(S_G) + \overline{z}(S_H) \Leftrightarrow d_l(C,C')$$

The constants $d_l(C,C')$ are called the extension constants *for $\overline{z}$.*

Note that, if there is a refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ for each $l \geq 0$ and there is an extension $\overline{z}$ of z with respect to $\{\sim_{rQ,l} \mid l \geq 0\}$, then it is not necessarily the case that $\overline{z}$ is an extension of z with respect to $\{\sim_{Q,l} \mid l \geq 0\}$. However, $\overline{z}$ is an extension for z with respect to any refinement of $\sim_{rQ,l}$.

Lemma 5.1. *Suppose $\overline{z}$ is an extension of z with respect to $\{\sim_{rQ,l} \mid l \geq 0\}$. Let G be an l-terminal graph ($l \geq 0$). Let $S, S' \in D_{[]}(G)$ such that $(G,S) \sim_{rQ,l} (G,S')$. For each terminal graph H and each $S_H \in D_{[]}(H)$, if $Q(G \oplus H, S \oplus S_H)$ holds, then*

$$z(S \oplus S_H) \Leftrightarrow z(S' \oplus S_H) \quad = \quad \overline{z}(S) \Leftrightarrow \overline{z}(S').$$

Proof. Let $C = \mathrm{ec}_{rQ,l}(G,S)$ and let d_l denote the extension constants for $\overline{z}$. Let H be a terminal graph and let $S_H \in D_{[]}(H)$ such that $Q(G \oplus H, S \oplus S_H)$ holds. Let $C' = \mathrm{ec}_{rQ,l}(H,S_H)$. Then $Q(G \oplus H, S' \oplus S_H)$ also holds. Furthermore, $z(S \oplus S_H) \Leftrightarrow z(S' \oplus S_H) = (\overline{z}(S) + \overline{z}(S_H) \Leftrightarrow d_l(C,C')) \Leftrightarrow (\overline{z}(S') + \overline{z}(S_H) \Leftrightarrow d_l(C,C')) = \overline{z}(S) \Leftrightarrow \overline{z}(S')$. □

In other words, Lemma 5.1 shows that if $(G,S) \sim_{rQ,l} (G,S')$ and $\overline{z}(S) > \overline{z}(S')$, then S always leads to better solutions than S' (assuming $\mathrm{opt} = \max$).

Let G be an l-terminal graph, and $C \in \mathcal{C}_{rQ,l}$. Let

$$\mathrm{opt}(G,C) \quad = \quad \mathrm{opt}\{\overline{z}(S) \mid S \in D_{[]}(G) \wedge \mathrm{ec}_{rQ,l}(G,S) = C\}$$

(hence $\mathrm{opt}(G,C) = \mathsf{false}$ if there is no $S \in D_{[]}(G)$ for which $\mathrm{ec}_{rQ,l}(G,S) = C$). If $\mathrm{opt}(G,C) \in \mathbb{Z}$, then let $\mathrm{optS}(G,C)$ denote an $S \in D_{[]}(G)$ for which $\overline{z}(S) = \mathrm{opt}(G,C)$. Informally speaking, $\mathrm{opt}(G,C)$ represents 'the value of the best partial solution of G in equivalence class C', and $\mathrm{optS}(G,C)$ gives such a partial solution (if existing).

Let $S \in D_{[]}(G)$, let $C = \mathrm{ec}_{rQ,l}(G,S)$ and suppose S may lead to an optimal solution, i.e. there is a terminal graph H and an $S_H \in D_{[]}(H)$ such that $Q(G \oplus H, S \oplus S_H)$ holds and $z(S \oplus S_H) = \Phi(G \oplus H)$. Lemma 5.1 shows that $\overline{z}(S) = \mathrm{opt}(G,C)$. Hence only partial solutions S for which $\overline{z}(S) = \mathrm{opt}(G, \mathrm{ec}_{rQ,l}(G,S))$ may lead to optimal solutions.

Theorem 5.2. *Let Φ be a constructive optimization problem defined by (D,Q,z,opt). Suppose D is a vertex-edge-tuple and there is a refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ for which the following conditions hold.*

1. *for each $l \geq 0$, $|\mathcal{C}_{rQ,l}|$ is finite.*
2. *There is an extension $\overline{z}$ of z with respect to $\{\sim_{rQ,l} \mid l \geq 0\}$ and for each $l \geq 0$, there is a constant $K_l \in \mathbb{N}$, such that for each l-terminal graph G and every $S, S' \in D_{[]}(G)$, if both S and S' can lead to optimal solutions, then $|\overline{z}(S) \Leftrightarrow \overline{z}(S')| \leq K_l$.*

Then for each $k \geq 1$, there exists a special constructive reduction-counter system $\mathcal{S}$ for Φ_k defined by $(D, Q_k, z, \mathrm{opt})$, and for each reduction-counter rule $((H_1, H_2), i)$ in $\mathcal{S}$, $H_1 \approx_{rQ,l} H_2$.

If, in addition, (i) Q and $\sim_{rQ,l}$ are effectively decidable, (ii) z is effectively computable, and (iii) in condition 2, $\overline{z}$ and K_l are effectively computable, then such a special constructive reduction-counter system can be effectively constructed.

Proof. Suppose conditions 1 and 2 hold for Φ. Let $\overline{z}$ be the extension of condition 2 and let $d_l(C, C')$ denote the corresponding extension constants for all $C, C' \in \mathcal{C}_{rQ,l}$. For each $l \geq 0$, let $K_l \in \mathbb{N}$ be as in condition 2. Let P be the construction property derived from Φ (i.e. P is defined by (D, Q)).

We first construct a refinement $\sim_l$ of $\sim_{rQ,l}$ such that for each pair (G_1, G_2) of l-terminal graphs, if $|V(G_2)| < |V(G_1)|$ and $G_1 \approx_l G_2$, then there is an $i \in \mathbb{Z}$ for which the following holds.

a. $((G_1, G_2), i)$ is a safe reduction-counter rule for Φ, and
b. for each $S_2 \in D_{[]}(G_2)$ which can lead to an optimal solution, there is an $S_1 \in D_{[]}(G_1)$ such that $(G_1, S_1) \sim_l (G_2, S_2)$ and for each l-terminal graph H and each $S \in D_{[]}(H)$, if $Q(G_2 \oplus H, S_2 \oplus S)$ holds and $z(S_2 \oplus S) = \Phi(G_2 \oplus H)$, then $Q(G_1 \oplus H, S_1 \oplus S)$ holds, and $z(S_1 \oplus S) = \Phi(G_1 \oplus H)$.

We also show that $\sim_l$ is finite. After that, we show how to use $\sim_l$ to build a special constructive reduction-counter system for Φ_k ($k \geq 1$).

For each $l \geq 0$, each l-terminal graph G, do the following. If there is a partial solution in G which can lead to an optimal solution, then let $\tilde{S}_G \in D_{[]}(G)$ such that $\tilde{S}_G$ can lead to an optimal solution. Let $i_G = \overline{z}(\tilde{S}_G)$ (note that $i_G \in \mathbb{Z}$). Otherwise, $\tilde{S}_G$ is not defined and $i_G = 0$. Let $h_G : \mathcal{C}_{rQ,l} \to \{\Leftrightarrow K_l, \ldots, K_l\} \cup \{\mathsf{false}\}$ be a function with for each $C \in \mathcal{C}_{rQ,l}$,

$$h_G(C) = \begin{cases} \mathrm{opt}(G, C) \Leftrightarrow i_G & \text{if } |\,\mathrm{opt}(G, C) \Leftrightarrow i_G| \leq K_l \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

For each $l \geq 0$, each pair G_1, G_2 of l-terminal graphs and each $S_1 \in D_{[]}(G_1)$ and $S_2 \in D_{[]}(G_2)$, let

$$\begin{aligned} (G_1, S_1) \sim_l (G_2, S_2) \quad \Leftrightarrow \quad & (G_1, S_1) \sim_{rQ,l} (G_2, S_2) \\ & \wedge h_{G_1}(ec_{rQ,l}(G_1, S_1)) = h_{G_2}(ec_{rQ,l}(G_2, S_2)). \end{aligned}$$

Note that $\sim_l$ is a refinement of $\sim_{rQ,l}$ and hence of $\sim_{Q,l}$. For each $l \geq 0$, the range of h_G for any l-terminal graph G has finite cardinality, and $\sim_{rQ,l}$ is finite, which means that $\sim_l$ is also finite.

Consider the equivalence relation $\approx_l$ on l-terminal graphs as defined in Definition 3.7. Let $l \geq 0$, let G_1 and G_2 be l-terminal graphs, such that $|V(G_2)| < |V(G_1)|$ and $G_1 \approx_l G_2$. By definition of $\sim_l$ and $\approx_l$, $h_{G_1} = h_{G_2}$. Let $i = i_{G_1} \Leftrightarrow i_{G_2}$, and let $h = h_{G_1} = h_{G_2}$. We show that G_1, G_2 and i satisfy conditions a and b given above.

Note that, if there is an $S \in D_{[]}(G_1)$ which can lead to a solution, then there is an $S' \in D_{[]}(G_2)$ which can lead to a solution, and vice versa.

Claim 5.1. Suppose there is a partial solution in G_1 which can lead to a solution. Let $C \in \mathcal{C}_{rQ,l}$ such that $\text{opt}(G_1,C) \in \mathbb{Z}$. Let H be an l-terminal graph. Let $S_1 = \text{optS}(G_1,C)$, $S_2 = \text{optS}(G_2,C)$ and $S_H \in D_{[]}(H)$, and suppose $Q(G_1 \oplus H, S_1 \oplus S_H)$ holds. Then $z(S_1 \oplus S_H) = z(S_2 \oplus S_H) + i$.

Proof. As there is a partial solution in G_1 which can lead to a solution, $\tilde{S}_{G_1}$ is defined and $\overline{z}(\tilde{S}_{G_1}) = i_{G_1}$. This also means that $\tilde{S}_{G_2}$ is defined and $\overline{z}(\tilde{S}_{G_2}) = i_{G_2}$. Hence, by condition 2 of the theorem, $|\overline{z}(S_1) \Leftrightarrow i_{G_1}| \leq K_l$, so $\overline{z}(S_1) = i_{G_1} + h(C)$, and similarly, $\overline{z}(S_2) = i_{G_2} + h(C)$. Furthermore,

$$\begin{aligned}
z(S_1 \oplus S_H) &= \overline{z}(S_1) + \overline{z}(S_H) \Leftrightarrow d_l(C,C') \\
&= h(C) + i_{G_1} + \overline{z}(S_H) \Leftrightarrow d_l(C,C') \\
&= h(C) + i_{G_2} \Leftrightarrow i_{G_2} + i_{G_1} + \overline{z}(S_H) \Leftrightarrow d_l(C,C') \\
&= \overline{z}(S_2) + \overline{z}(S_H) \Leftrightarrow d_l(C,C') \Leftrightarrow i_{G_2} + i_{G_1} \\
&= z(S_2 \oplus S_H) \Leftrightarrow i_{G_2} + i_{G_1} \\
&= z(S_2 \oplus S_H) + i.
\end{aligned}$$

□

Claim 5.2. $((G_1,G_2),i)$ is safe for Φ.

Proof. Let H be an l-terminal graph. We have to show that $\Phi(G_1 \oplus H) = \Phi(G_2 \oplus H) + i$. Since $G_1 \approx_l G_2$, and $\approx_l$ is a refinement of $\approx_{Q,l}$, which in turn is a refinement of $\sim_{P,l}$, $\Phi(G_1 \oplus H)$ is false if and only if $\Phi(G_2 \oplus H)$ is false. Hence if $\Phi(G_1 \oplus H) = \text{false}$, then $\Phi(G_1 \oplus H) = \Phi(G_2 \oplus H) + i$.

Now suppose $\Phi(G_1 \oplus H) \in \mathbb{Z}$, and let $S \in D(G_1 \oplus H)$ such that $z(S) = \Phi(G_1 \oplus H)$. Let $S_1 = S[G_1]$ and $S_H = S[H]$. Let $S_2 = \text{optS}(G_2, \text{ec}_{rQ,l}(G_1,S_1))$. By the previous claim, $z(S_1 \oplus S_H) = z(S_2 \oplus S_H) + i$, and hence if opt $=$ max, then $\Phi(G_1 \oplus H) \leq \Phi(G_2 \oplus H) + i$, and if opt $=$ min, then $\Phi(G_1 \oplus H) \geq \Phi(G_2 \oplus H) + i$. By symmetry, we can also show that if opt $=$ max, then $\Phi(G_2 \oplus H) \leq \Phi(G_1 \oplus H) \Leftrightarrow i$ and if opt $=$ min then $\Phi(G_2 \oplus H) \geq \Phi(G_1 \oplus H) \Leftrightarrow i$, and hence $\Phi(G_1 \oplus H) = \Phi(G_2 \oplus H) + i$. □

Claim 5.3. For each $S_2 \in D_{[]}(G_2)$ which can lead to an optimal solution, there is an $S_1 \in D_{[]}(G_1)$ such that $(G_1,S_1) \sim_l (G_2,S_2)$ and for each l-terminal graph H and each $S \in D_{[]}(H)$, if $Q(G_2 \oplus H, S_2 \oplus S)$ holds and $z(S_2 \oplus S) = \Phi(G_2 \oplus H)$, then $Q(G_1 \oplus H, S_1 \oplus S)$ holds, and $z(S_1 \oplus S) = \Phi(G_1 \oplus H)$.

Proof. Let $S_2 \in D_{[]}(G_2)$ such that S_2 can lead to an optimal solution, let $C = \text{ec}_{rQ,l}(G_2,S_2)$. Note that $\text{opt}(G_2,C) = \overline{z}(S_2) \neq \text{false}$ (and hence $\text{opt}(G_1,C) \neq \text{false}$). Let $S_1 = \text{opt}(G_1,C)$. Let H be an l-terminal graph, let $S_H \in D_{[]}(H)$ and let $C' = \text{ec}_{rQ,l}(H)$. Suppose $Q(G_2 \oplus H, S_2 \oplus S_H)$

holds and $z(S_2 \oplus S_H) = \Phi(G_2 \oplus H)$. By a previous claim, $z(S_1 \oplus S_H) = z(S_2 \oplus S_H) + i$. Since $\Phi(G_1 \oplus H) = \Phi(G_2 \oplus H) + i$ and $\Phi(G_2 \oplus H) = z(S_2 \oplus H)$, this implies that $z(S_1 \oplus H) = \Phi(G_1 \oplus H)$. □

The claims show that conditions a and b hold.

Let $k \geq 1$. We show that there is a special constructive reduction-counter system for Φ_k. Theorem 3.2 shows that there is a special constructive reduction system $\mathcal{S} = (\mathcal{R}, \mathcal{I}, A_{\mathcal{R}}, A_{\mathcal{I}})$ for P_k such that for each $(H_1, H_2) \in \mathcal{R}$, $H_1 \approx_l H_2$. We show how to transform $\mathcal{S}$ into a special constructive reduction-counter system $\mathcal{S}' = (\mathcal{R}', \mathcal{I}', \phi, A'_{\mathcal{R}}, A'_{\mathcal{I}})$ for Φ_k. First, we make a set $\mathcal{R}'$ of reduction-counter rules from $\mathcal{R}$: for each $r = (H_1, H_2) \in \mathcal{R}$, make a reduction-counter rule (r, i) in $\mathcal{R}'$ with $i = i_{H_1} - i_{H_2}$. By condition a, $\mathcal{R}'$ is safe for Φ_k.

Next, let $\mathcal{I}' = \mathcal{I}$, and for each $G \in \mathcal{I}'$, let $\phi(G) = \Phi(G)$. We let the algorithms $A'_{\mathcal{R}}$ and $A'_{\mathcal{I}}$ be the same as $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$, but with different tables. For $A'_{\mathcal{I}}$, we make a table which gives for each $G \in \mathcal{I}'$ an $S \in D(G)$ such that $\Phi(G) = z(S)$. For $A'_{\mathcal{R}}$, we make a table which, for each reduction-counter rule $r = ((H_1, H_2), i) \in \mathcal{R}'$, and each $S_2 \in H_2$ for which $\overline{z}(S_2) = \text{opt}(H_2, \text{ec}_l(H_2, S_2))$, contains $\text{optS}(H_1, S_1)$. Now, $(\mathcal{R}', \mathcal{I}', \phi, A'_{\mathcal{R}}, A'_{\mathcal{I}})$ is a special constructive reduction-counter system for Φ_k. The effectiveness result easily follows. □

Note that, if only condition 1 of Theorem 5.2 holds for Φ, then Φ is of finite integer index, and hence for each $k \geq 1$, there is a special reduction-counter system for Φ_k.

In the following theorem we show for a number of constructive optimization problems that they are efficiently solvable, using the methods of Theorem 5.2.. The proofs are all of the same type; we only give the first one completely, the others can be found in [11].

Theorem 5.3. *Each of the following constructive optimization problems can be solved in $O(n)$ time and space on graphs of bounded treewidth without making a tree decomposition of the input graph.*

MAX INDUCED d-DEGREE SUBGRAPH for all $d \geq 0$: *given a graph G, find a set $W \subseteq V(G)$ of maximum cardinality such that the degree of each vertex in $G[W]$ is at most d (for $d = 0$ this is* MAX INDEPENDENT SET*).*

MIN VERTEX COVER: *given a graph G, find a set $W \subseteq V(G)$ of minimum cardinality, such that each edge in G has at least one end point in W.*

MIN p-DOMINATING SET for all $p \geq 1$: *given a graph G, find a set $W \subseteq V(G)$ of minimum cardinality such that each $v \in V(G) - W$ has at least p neighbors in W.*

MAX CUT on graphs with bounded degree.

MIN PARTITION INTO CLIQUES: *given a graph G, find a partition $\{V_1, \ldots, V_s\}$ of $V(G)$ such that s is minimum and for each i, $G[V_i]$ is a complete graph.*

MIN HAMILTONIAN PATH COMPLETION: *given a graph G, find the minimum number of edges that should be added to G such that G contains a Hamiltonian path.*

MIN HAMILTONIAN CIRCUIT COMPLETION: *given a graph G, find the minimum number of edges that should be added to G such that G contains a Hamiltonian cycle.*

MAX LEAF SPANNING TREE: *given a graph G, find a spanning tree of G in which the number of leaves is maximum.*

Proof. For each $l \geq 0$, let $I_l = \{1, \dots, l\}$, and $F_l = \{\{i, j\} \mid 1 \leq i < j \leq l\}$. Furthermore, for each l-terminal graph $G = (V, E, \langle x_1, \dots, x_l \rangle)$, let

$$F(G) = \{\{i, j\} \mid \{x_i, x_j\} \in E\},$$

and for each $W \subseteq V(G)$ let

$$I(W) = \{i \in I_l \mid x_i \in W\}.$$

We give the full proof for MAX INDUCED d-DEGREE SUBGRAPH; for the other problems we omit many (lengthy) details.

MAX INDUCED d-DEGREE SUBGRAPH. Let $d \geq 0$ be fixed. Let Φ be defined by $(D, Q, z, \max)$, where D, Q and z are defined as follows. For each graph G, let $D(G) = \mathcal{P}(V)$, and for each $S \in D(G)$, let

$$Q(G, S) = \text{'for all } v \in S\text{: } |N_{G,S}(v)| \leq d\text{'},$$

where $N_{G,S}(v) = \{w \in S \mid \{v, w\} \in E(G)\}$. Furthermore, let $z(S) = |S|$. We show that for each $k \geq 1$, there is a special constructive reduction-counter system for Φ_k, by using Theorem 5.2. We define a refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ by giving the sets $C_{rQ,l}$ and the functions $\mathrm{ec}_{rQ,l}$. For each $l \geq 0$, let

$$\begin{aligned} C_{rQ,l} = \ & \{(I, \mathsf{false}) \mid I \subseteq I_l\} \cup \\ & \{(F, I, N) \mid F \subseteq F_l \wedge I \subseteq I_l \wedge N \subseteq \{(i, n) \mid i \in I \wedge n \in \{1, \dots, d\}\}. \end{aligned}$$

$|C_{rQ,l}|$ is bounded, because d is fixed. For each l-terminal graph $G = (V, E, \langle x_1, \dots, x_l \rangle)$, each $S \in D_{[]}(G)$, let $\mathrm{ec}_{rQ,l}(G, S) \in C_{rQ,l}$ be defined as follows. If there is a $v \in S$ such that $|N_{G,S}(v)| > d$, then $\mathrm{ec}_{rQ,l}(G, S) = (I(S), \mathsf{false})$ (S can not lead to a solution), otherwise, $\mathrm{ec}_{rQ,l}(G, S) = (F(G), I(S), N)$, where $N = \{(i, |N_{G,S}(x_i)|) \mid i \in I(S)\}$.

We first show that $\sim_{rQ,l}$ is a refinement of $\sim_{Q,l}$ for all l. Suppose $(G_1, S_1) \sim_{rQ,l} (G_2, S_2)$. Clearly, (G_1, S_1) and (G_2, S_2) are compatible. Let H be an l-terminal graph, let $S_H \in D_{[]}(H)$ such that (G_1, S_1) and (H, S_H) are $\oplus$-compatible. We have to show that $Q(G_1 \oplus H, S_1 \oplus S_H)$ holds if and only if $Q(G_2 \oplus H, S_2 \oplus H)$ holds. If $\mathrm{ec}_{rQ,l}(G_1, S_1) = \mathrm{ec}_{rQ,l}(G_2, S_2) = (I(S_1), \mathsf{false})$, then $Q(G_1 \oplus H, S_1 \oplus S_H) = \mathsf{false} = Q(G_2 \oplus H, S_2 \oplus S_H)$.

Suppose $\mathrm{ec}_{rQ,l}(G_1, S_1) = \mathrm{ec}_{rQ,l}(G_2, S_2) = (F, I, N)$, where $N = \{(i, n_i) \mid i \in I\}$. Let $X = \langle x_1, \dots, x_l \rangle$, $Y = \langle y_1, \dots, y_l \rangle$, and $Z = \langle z_1, \dots, z_l \rangle$ denote the terminal sets of G_1, G_2 and H, respectively.

$$\begin{aligned} & Q(G_1 \oplus H, S_1 \oplus S_H) \\ & \quad = (\forall_{v \in S_1 \oplus S_H} |N_{G_1 \oplus H, S_1 \oplus S_H}(v)| \leq d) \\ & \quad = (\forall_{i \in I} |N_{H,S_H}(z_i)| + |N_{G_1,S_1}(x_i)| \Leftrightarrow |\{j \in I \mid x_j \in N_{G_1,S_1}(x_i) \wedge z_j \in N_{H,S_H}(z_i)\}| \leq d) \\ & \qquad \wedge (\forall_{v \in S_1 - X} |N_{G_1,S_1}(v)| \leq d) \wedge (\forall_{v \in S_H - Z} |N_{H,S_H}(v)| \leq d) \\ & \quad = (\forall_{i \in I} |N_{H,S_H}(z_i)| + |n_i| \Leftrightarrow |\{j \in I \mid \{i, j\} \in F \wedge \{z_i, z_j\} \in E(H)\}| \leq d) \\ & \qquad \wedge (\forall_{v \in S_1 - X} |N_{G_1,S_1}(v)| \leq d) \wedge (\forall_{v \in S_H - Z} |N_{H,S_H}(v)| \leq d) \\ & \quad = (\forall_{i \in I} |N_{H,S_H}(z_i)| + |N_{G,S_2}(y_i)| \Leftrightarrow |\{j \in I \mid y_j \in N_{G_2,S_2}(y_i) \wedge z_j \in N_{H,S_H}(z_i)\}| \leq d) \\ & \qquad \wedge (\forall_{v \in S_2 - Y} |N_{G_2,S_2}(v)| \leq d) \wedge (\forall_{v \in S_H - Z} |N_{H,S_H}(v)| \leq d) \\ & \quad = Q(G_2 \oplus H, S_2 \oplus S_H) \end{aligned}$$

Hence $\sim_{rQ,l}$ is a refinement of $\sim_{Q,l}$. This proves condition 1 of Theorem 5.2.

Consider condition 2 of Theorem 5.2. For each terminal graph G, each $S \in D_{[]}(G)$, let $\overline{z}(S) = |S|$. We show that $\overline{z}$ is an extension of z. Let $C, C' \in \mathcal{C}_{rQ,l}$, such that C and C' are compatible. Let $I \subseteq I_l$ such that $C = (I, \mathsf{false})$ or $C = (F, I, N)$ for some F and N, and $C' = (I, \mathsf{false})$ or $C' = (F', I, N')$ for some F' and N'. Let G and H be l-terminal graphs, let $S \in D_{[]}(G)$ and $S' \in D_{[]}(H)$ such that $\mathrm{ec}_{rQ,l}(G,S) = C$ and $\mathrm{ec}_{rQ,l}(H,S') = C'$. Then $z(S \oplus S') = |S \oplus S'| = |S \cup S'| = |S| + |S'| \Leftrightarrow |I| = \overline{z}(S) + \overline{z}(S') \Leftrightarrow |I|$, hence $d_l(C,C') = |I|$, which shows that $\overline{z}$ is an extension of z.

For each $l \geq 0$ let $K_l = 2l$. Let G be an l-terminal graph. Let $C_G \in \mathcal{C}_{rQ,l}$ denote the equivalence class $(F(G), \emptyset, \emptyset)$. Note that $\mathrm{opt}(G, C_G) \neq \mathsf{false}$, since $\mathrm{ec}_{rQ,l}(G, \emptyset) = C_G$.

Let $S \in D_{[]}(G)$. We show that, if S can lead to an optimal solution, then $|\overline{z}(S) \Leftrightarrow \mathrm{opt}(G, C_G)| \leq l$. This proves that condition 2 of Theorem 5.2 with $K_l = 2l$.

Claim 5.4. If S can lead to an optimal solution then $|\overline{z}(S) \Leftrightarrow \mathrm{opt}(G, C_G)| \leq l$.

Proof. Suppose S can lead to an optimal solution.

First consider the value of $\overline{z}(S) \Leftrightarrow \mathrm{opt}(G, C_G)$. Let $S' = S \Leftrightarrow X$. Note that $\mathrm{ec}_{rQ,l}(G, S') = C_G$ and $\overline{z}(S) \leq \overline{z}(S') + l$. Hence $\overline{z}(S) \Leftrightarrow \mathrm{opt}(G, C_G) \leq \overline{z}(S') + l \Leftrightarrow \mathrm{opt}(G, C_G) \leq l$.

Next consider the value of $\mathrm{opt}(G, C_G) \Leftrightarrow \overline{z}(S)$. Suppose that $\mathrm{opt}(G, C_G) \Leftrightarrow \overline{z}(S) > l$. Let H be an l-terminal graph and $S_H \in D_{[]}(H)$ such that (G,S) and (H, S_H) are $\oplus$-compatible and $S \oplus S_H$ is an optimal solution of $G \oplus H$ (this is possible since S can lead to an optimal solution). Let $S' \in D(G \oplus H)$ be the set obtained from $S \oplus S_H$ by deleting all terminals from G. Note that $Q(G \oplus H, S')$ holds, and thus $\mathrm{ec}_{rQ,l}(G, S'[G]) = C_G$. Furthermore $z(S') \geq z(S \oplus S_H) \Leftrightarrow l$. But then $\mathrm{optS}(G, C_G) \oplus S'[H]$ is also a solution for $G \oplus H$, and furthermore,

$$\begin{aligned} z(\mathrm{optS}(G, C_G) \oplus S'[H]) &= \overline{z}(\mathrm{optS}(G, C_G)) + \overline{z}(S'[H]) \\ &> \overline{z}(S'[G]) + l + \overline{z}(S'[H]) \\ &= z(S') + l \\ &\geq z(S \oplus S_H). \end{aligned}$$

This is a contradiction, since $S \oplus S_H$ is an optimal solution. Hence $\mathrm{opt}(G, C_G) \Leftrightarrow \overline{z}(S) \leq l$. □

This proves that conditions 1 and 2 of Theorem 5.2 hold. Moreover, Q and $\sim_{rQ,l}$ are effectively decidable and $\overline{z}$ and K_l are effectively computable, and thus there is an effectively computable special constructive reduction-counter system for MAX INDUCED d-DEGREE SUBGRAPH.

MIN VERTEX COVER and MIN p-DOMINATING SET. Similar to MAX INDUCED d-DEGREE SUBGRAPH.

MAX CUT on graphs with bounded degree and MAX LEAF SPANNING TREE . Can be solved with techniques, similar to the other problems considered here.

MIN PARTITION INTO CLIQUES. Note that partitions $\{V_1, \ldots, V_s\}$ of the vertices of a graph can not be directly represented by a vertex-edge-tuple, since the number s can be arbitrarily large, depending on the size of the graph. Therefore, we define for each graph G, $D(G) = P(E(G))$, and for each $S \in D(G)$, we let

$$Q(G,S) = \text{'each component of } (V(G), S) \text{ is a clique'},$$

opt $=$ min and $z(S) =$ 'the number of components of $(V(G),S)$'. Given a set $F \subseteq E(G)$ for which $Q(G,F)$ holds, we can compute a clique partition of G by computing the connected components of $(V(G),F)$. This can be done in linear time.

Alternatively, we can, during the construction phase of the reduce-construct algorithm, maintain a clique partition of the current graph as a set of subsets of the vertices. This can be done in a similar way as for vertex-edge-tuples. The remaining details of the proof are omitted.

MIN HAMILTONIAN PATH COMPLETION. We have the same problem as for MIN PARTITION INTO CLIQUES: a set of 'edges to be added' can not be represented by a vertex-edge-tuple. Therefore, we define the problem as follows. Let Φ be defined by $(D,Q,z,\min)$, where D, Q and z are defined as follows. For each graph G, let $D(G)$ be a set of edge sets $F \subseteq E(G)$ for which each component of $(V(G),F)$ is a path. For each $S \in D(G)$, let $Q(G,S) =$ true, and let $z(S) =$ 'the number of components of $(V(G),S)$'.

Again we can compute such a set of extra edges in linear time from an optimal solution F for G: compute the components of $(V(G),F)$, and for each such component, find the end vertices of the path. Now concatenate the paths in an arbitrary way. The edges that are added for the concatenation are the desired edges. The remaining details of the proof are omitted.

MIN HAMILTONIAN CIRCUIT COMPLETION. Use the algorithm for MIN HAMILTONIAN PATH COMPLETION: if a graph is not Hamiltonian (which can be tested a reduction algorithm as HAMILTONIAN CIRCUIT is MS-definable), then its Hamiltonian circuit completion number is one larger than its Hamiltonian path completion number. □

6 Parallel Reduction Algorithms

In [8] an efficient parallel variant of algorithm Reduce was given, based on a variant of the special reduction system. In this section we show how to use this algorithm to make an efficient parallel variant of algorithm Reduce-Construct (Section 6.2). We also show that the parallel variant of Theorem 3.2 holds. Furthermore we show how to extend the parallel algorithm such that it can also be used for (constructive) optimization problems (Sections 6.3 and 6.4), and we give the parallel variants of Theorems 4.2 and 5.2. We show that these algorithms can be used for large classes of problems on graphs of small treewidth.

We start with a description of the parallel reduction algorithm as introduced in [8].

6.1 Decision Problems

The basic idea of the parallel reduction algorithm is that, if there are two or more possible applications of reduction rules at a certain time, and these applications do not interfere, then they can be applied concurrently.

Definition 6.1 (Non-Interfering Matches). *Let $\mathcal{R}$ be a set of reduction rules and let G be a graph with a fixed adjacency list representation. Two matches G_1 and G_2 in G are said to be* non-interfering *if*

- *no inner vertex of G_i ($i = 1,2$) is a vertex of G_{3-i},*

- *the sets of edges of G_1 and G_2 are disjoint, and*
- *if G_1 and G_2 have a common terminal x, then in the adjacency list of x, there are no two consecutive edges e_1 and e_2 such that $e_1 \in E(G_1)$ and $e_2 \in E(G_2)$.*

A set of matches in G is non-interfering if all matches in the set are pairwise non-interfering.

Let $\mathcal{R}$ be a set of reduction rules and let G be a graph with a fixed adjacency list representation. If we have a set of non-interfering matches in G, then the reductions corresponding to these matches can be executed in parallel without concurrent reading or writing, and this gives the same result as if the reductions were executed subsequently, in an arbitrary order. In order to make an efficient parallel reduction algorithm for a given graph property P, we want to have a special reduction system which gives sufficiently many matches in any graph G for which P holds. Therefore, we introduce a special *parallel* reduction system.

Definition 6.2 (Special Parallel Reduction System). *Let P be a graph property, and $(\mathcal{R}, \mathcal{I})$ a reduction system for P. Let n_{max} be the maximum number of vertices in any left-hand side of a rule $r \in \mathcal{R}$. $(\mathcal{R}, \mathcal{I})$ is called a* special parallel reduction system *for P if we know positive integers n_{min} and d, $n_{min} \leq n_{max} \leq d$, and a constant $c > 0$, such that the following conditions hold.*

1. *For each reduction rule $(H_1, H_2) \in \mathcal{R}$, H_1 and H_2 are open and connected.*
2. *For each connected graph G and each adjacency list representation of G, if $P(G)$ holds and G has at least n_{min} vertices, then G contains at least $c \cdot |V(G)|$ d-discoverable matches.*

Note that, since for each integer $n > 1$ and each constant c, if $c > 0$ then $cn > 0$, a special parallel reduction system is also a special reduction system.

Consider the graph property which holds if a graph is a two-colorable cycle. The reduction system that we have given for this property in Figure 3 is an example of a special parallel reduction system (take $d = n_{max} = n_{min} = 5$ and $c = 1/5$).

Let P be a graph property and $S = (\mathcal{R}, \mathcal{I})$ a special parallel reduction system for P. Let n_{min}, n_{max}, d and c be as in Definition 6.2. The parallel reduction algorithm introduced in [8] based on S works as follows. The algorithm finds d-discoverable matches and executes the corresponding reductions, until there are no more d-discoverable matches. In more detail, the following is done.

Suppose we are given an input graph G with n vertices. The algorithm consists of a number of reduction rounds, which are executed subsequently. In each reduction round, $\Omega(m)$ reductions are applied to the current graph, which has m vertices, if $P(G)$ holds. This is done in four steps.

1. In the first step, the algorithm finds a d-discoverable match from each vertex v which has degree at most d and is an inner vertex of a d-discoverable match. If this succeeds, the corresponding reduction rule r is looked up. Let A denote the set of all matches that are found. Note that A is not necessarily non-interfering.
2. In the second step, the algorithm computes a subset A' of A with size $\Omega(|A|)$, which is a set of non-interfering matches.

3. In the last step, all reductions corresponding to the matches in A' are applied.

The first and third step can be done in constant time on m processors, without concurrent reading or writing: in step 1, take one processor for each vertex of degree at most d. In step 3, for each match in A', let the processor which discovered the match in step 2 apply its corresponding reduction. The second step is more complicated. It is basically done as follows. First, a *conflict graph* of all matches in A is built. This graph contains a vertex for each match in A, and an edge between two vertices if and only if the corresponding matches are interfering. Now an independent set in the conflict graph corresponds to a set of non-interfering matches. It can be seen that the conflict graph has bounded degree. This means that there is an independent set A' of size $\Omega(|A|)$ which can be found efficiently in parallel on an EREW PRAM (for more details, see [8]).

Note that in step 2, the size of A is at least cm as long as P holds for the input graph. This implies that at most $O(\log n)$ reduction rounds have to be done: if the graph resulting after these steps is in $\mathcal{I}$, then P holds for the input graph and true is returned. Otherwise, P does not hold for the input graph and false is returned.

Consider the amount of resources used by the algorithms. As said before, we have $O(\log n)$ reduction rounds, and in each reduction round the number of vertices of the graph is reduced by a constant fraction (if P holds for the input graph). The only part in a reduction round which takes more than constant time is step 2. By a careful analysis, it can be seen that the algorithm can be made to run in $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM. For a CRCW PRAM, the algorithm can be slightly improved: it runs in $O(\log n)$ time with $O(n)$ operations and space (see [8] for details).

Theorem 6.1. *Let P be a graph property. If we have a special parallel reduction system for P, then we have an algorithm which decides P on connected graphs in $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM, and in $O(\log n)$ time with $O(n)$ operations and space on a CRCW PRAM.*

The definition of a special parallel reduction system, Lemma 2.6 and (the proof of) Theorem 2.2 immediately imply the following result.

Theorem 6.2. *Let P a graph property, and suppose P is of finite index. For each integer $k \geq 1$, there is a special parallel reduction system for P_k.*

If P is also effectively decidable, and there is an equivalence relation $\sim_l$ for each $l \geq 0$, which is a finite refinement of $\sim_{P,l}$ and is effectively decidable, then such a system $(\mathcal{R}, \mathcal{I})$ can effectively be constructed.

The analog of Corollary 2.2 also holds for the parallel case.

In the parallel case, there exist algorithms that decide finite index properties in $O(\log n)$ time with $O(n)$ operations and space, given a tree decomposition of bounded width of the graph [15]. However, the best known parallel algorithm for finding a tree decomposition of the input graph takes $O(\log^2 n)$ time with $O(n)$ operations on an EREW or CRCW PRAM [8]. Hence the reduction algorithms presented in this section are more efficient.

6.2 Construction Problems

We start with adapting the definition of a special constructive reduction system.

Definition 6.3. *Let P be a construction property defined by (D,Q) and let $(\mathcal{R},\mathcal{I},A_{\mathcal{R}},A_{\mathcal{I}})$ be a constructive reduction system for P. Algorithm $A_{\mathcal{R}}$ is* non-interfering *if for each graph G, each $S \in D(G)$, if $A_{\mathcal{R}}$ is executed simultaneously for the reconstructions corresponding to the undoing of two non-interfering reductions, then this gives the same result as running $A_{\mathcal{R}}$ successively for these two reconstructions. Furthermore, no concurrent reading or writing takes place.*

Definition 6.4 (Special Parallel Constructive Reduction System). *Let P be a construction property defined by (D,Q). A constructive reduction system $S = (\mathcal{R},\mathcal{I},A_{\mathcal{R}},A_{\mathcal{I}})$ for P is a* special parallel constructive reduction system *for P if*

- *$(\mathcal{R},\mathcal{I})$ is a special parallel reduction system for P,*
- *algorithms $A_{\mathcal{R}}$ and $A_{\mathcal{I}}$ use $O(1)$ time on a single processor, and*
- *algorithm $A_{\mathcal{R}}$ is non-interfering.*

Note that the constructive reduction system that we have defined for two-colorability of cycles (Figure 4) is a special parallel constructive reduction system: we represent each two-coloring as a labeling of the graph, i.e. each vertex is labeled with an integer denoting its color. We can implement algorithm $A_{\mathcal{R}}$ such that it is non-interfering, and it runs in $O(1)$ time (use the tables as given in Figure 4). Algorithm $A_{\mathcal{I}}$ also takes $O(1)$ time.

If we have a special parallel constructive reduction system for a given construction property P defined by (D,Q), then we can use a parallel variant of algorithm Reduce-Construct to construct a solution for an input graph G, if one exists. The parallel algorithm consists of two parts. In part one, reductions are applied as often as possible, using the parallel algorithm described in Section 6.1.

Part two of the algorithm starts with constructing an initial solution for the reduced graph, if P holds. This is done by one processor in constant time, by using algorithm $A_{\mathcal{I}}$. After that, the reduction rounds of part one are undone in reversed order. In each undo-action of a reduction round, all reductions of that round are undone, and the solution is adapted. Each undo-action of a reduction is executed by the same processor that applied the rule in the first part of the algorithm. This processor also applies algorithm $A_{\mathcal{R}}$. Since $A_{\mathcal{R}}$ is non-interfering, this results in the correct output.

Part one of the algorithm takes $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM. Part two can be done in $O(\log n)$ time with $O(n)$ operations and space on an EREW PRAM: each undo action of a reduction can be done in $O(1)$ time on one processor, and the local adaptation of the solution can also be done in $O(1)$ time by the same processor, since algorithm $A_{\mathcal{R}}$ takes constant time. This implies the following result.

Theorem 6.3. *Let P be a construction property defined by (D,Q). If we have a special parallel constructive reduction system for P, then we have an algorithm which, given a connected graph G, checks if $P(G)$ holds and if so, constructs an $S \in D(G)$ for which $Q(G,S)$ holds. The algorithm takes $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM, and $O(\log n)$ time with $O(n)$ operations and space on a CRCW PRAM.*

We next show that for a large class of construction properties on graphs of bounded treewidth, there is a special constructive reduction system.

Theorem 6.4. *Let P be a construction property defined by (D,Q). If D is a vertex-edge-tuple and $\sim_{Q,l}$ is finite for each $l \geq 0$, then for each $k \geq 1$, there is a special parallel constructive reduction system for P_k.*

If in addition, Q and a finite refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ are effectively decidable, then such a system can be effectively constructed.

Proof. Let $k \geq 1$. Let $S = (\mathcal{R}, \mathcal{I}, \mathcal{A}_{\mathcal{R}}, \mathcal{A}_{\mathcal{I}})$ be a special constructive reduction system for P_k as defined in the proof of Theorem 3.2. We show that $\mathcal{A}_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{I}}$ can be made such that S is a special parallel reduction system for P_k.

We use the following data structure for storing (partial) solutions. Suppose G is the current graph and $S = (S_1, S_2, \ldots, S_t)$ is the current solution for G. With each vertex v, we store booleans $b_1, \ldots, b_t$: for each i, $1 \leq i \leq t$, b_i is true if and only if $D_i(G) = V(G)$ and $v = S_i$, or $D_i(G) = \mathcal{P}(V(G))$ and $v \in S_i$. Similarly, with each edge e, we store booleans $b_1, \ldots, b_t$: for each i, $1 \leq i \leq t$, b_i is true if and only if $D_i(G) = E(G)$ and $e = S_i$, or $D_i(G) = \mathcal{P}(E(G))$ and $e \in S_i$. It is easy to see that with this data structure, we can make $\mathcal{A}_{\mathcal{R}}$ such that it is non-interfering and runs in $O(1)$ time. Furthermore, $\mathcal{A}_{\mathcal{I}}$ also runs in $O(1)$ time. □

Note that, with the data structure for t-vertex-edge-tuples as described in the proof of Theorem 6.4, a returned solution for a given input graph is represented as a labeling of the vertices and edges of the graph. However, we can transform this representation into the representation as described in the proof of Theorem 3.2: for each i, $1 \leq i \leq t$, use a parallel prefix algorithm (see e.g. [14]) to make a list of all vertices or edges for which b_i is true. Since t is fixed, this takes $O(\log n)$ time with $O(n)$ operations on an EREW PRAM, and hence does not increase the total running time.

In particular, Theorem 6.4 shows that many well-known graph problems, when restricted to graphs of bounded treewidth, can be solved constructively within the stated resource bounds. These include all MS-definable construction properties for which the domain is a vertex-edge-tuple.

6.3 Optimization Problems

It is again easy to adapt the parallel reduction algorithm for optimization problems. Therefore, we define a special parallel reduction-counter system to be a reduction-counter system of which the derived reduction system is a special parallel reduction system.

For instance, the reduction-counter system for MAX INDEPENDENT SET on cycles that we defined in Figure 5 is a special parallel reduction-counter system for this problem.

Let Φ be a graph optimization problem, and $S = (\mathcal{R}, \mathcal{I}, \phi)$ a special parallel reduction-counter system for Φ. A parallel reduction algorithm based on S is a combination of the parallel reduction algorithm based on the derived reduction system, and the sequential reduction algorithm described in Section 4. Each processor has a counter, which is initially set to zero. If a processor applies a reduction-counter rule in the algorithm, then it uses its own counter. After the last reduction round is finished, the counters of all processors are added up. Let *cnt* denote the resulting counter, let G denote the input graph and H the reduced graph. Now, if $H \in \mathcal{I}$, then $\Phi(G) = cnt + \phi(H)$, otherwise, $\Phi(G) = \Phi(H) = \mathsf{false}$. The sum of all the counters can be computed in $O(\log n)$ time with $O(n)$ operations and space on an EREW PRAM.

Theorem 6.5. *Let Φ be a graph optimization problem. If we have a special parallel reduction-counter system for Φ, then we have an algorithm which, for each connected graph G with n vertices, computes $\Phi(G)$ in $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM, and in $O(\log n)$ time with $O(n)$ operations and space on a CRCW PRAM.*

By Lemma 2.6 and Theorem 4.2, we also have the following result.

Theorem 6.6. *Let Φ be a graph optimization problem which is of finite integer index. For each integer $k \geq 1$, there exists a special parallel reduction system $\mathcal{S}$ for Φ_k.*

If, in addition, Φ is effectively computable, and there is an equivalence relation $\sim_l$, for each $l \geq 0$, which is a finite refinement of $\sim_{\Phi,l}$ and is effectively decidable, then such a system $\mathcal{S}$ can effectively be constructed.

Theorem 6.6 implies that there are special parallel reduction-counter systems for the following problems on graphs of bounded treewidth (see also Theorem 4.3): MAX INDUCED d-DEGREE SUBGRAPH for all $d \geq 0$, MIN p-DOMINATING SET for all $p \geq 1$, MIN VERTEX COVER, MAX CUT on graphs with bounded degree, MIN PARTITION INTO CLIQUES, MIN HAMILTONIAN PATH COMPLETION, and MAX LEAF SPANNING TREE.

6.4 Constructive Optimization Problems

A similar approach can be taken for constructive optimization problems. Let Φ be a constructive optimization problem defined by (D, Q, z, opt). Let $\mathcal{S}$ be a special constructive reduction-counter system for P. Then $\mathcal{S}$ is a *special parallel constructive reduction-counter system* if the derived constructive reduction system is a special parallel constructive reduction system.

Note that the constructive reduction-counter system that we defined for MAX INDEPENDENT SET on cycles (Figure 7) is a special parallel constructive reduction-counter system, if we represent an independent set as a labeling of the vertices of the graph: each vertex is labeled with a boolean which is true if and only if the vertex is in the independent set.

In the same way as described above we can transform the parallel algorithm for optimization problems as given in Section 6.3 into a parallel algorithm for constructive optimization problems, based on a special parallel constructive reduction-counter system.

Theorem 6.7. *Let Φ be a constructive optimization problem defined by (D, Q, z, opt). If we have a special parallel constructive reduction-counter system for Φ, then we have an algorithm which, given a connected graph G, checks if $\Phi(G) \in \mathbb{Z}$, and if so, constructs an $S \in D(G)$ for which $Q(G, S)$ holds and $z(S) = \Phi(G)$. The algorithm takes $O(\log n \log^* n)$ time with $O(n)$ operations and space on an EREW PRAM, and $O(\log n)$ time with $O(n)$ operations and space on a CRCW PRAM.*

From Theorem 6.4 and Theorem 5.2, we also derive the following result.

Theorem 6.8. *Let Φ be a constructive optimization problem defined by (D, Q, z, opt), where D is a vertex-edge-tuple. Suppose there is a refinement $\sim_{rQ,l}$ of $\sim_{Q,l}$ for which the following conditions hold.*

1. *For each $l \geq 0$, $|C_{rQ,l}|$ is finite.*
2. *There is an extension $\overline{z}$ with respect to $\{\sim_{rQ,l} | \ l \geq 0\}$ and for each $l \geq 0$, there is a constant $K_l \in \mathbb{N}$, such that for each for each l-terminal graph G, each $S, S' \in D_{[]}(G)$, if both S and S' can lead to optimal solutions, then $|\overline{z}(S) \Leftrightarrow \overline{z}(S')| \leq K_l$.*

Then for each $k \geq 1$, there exists a special parallel constructive reduction-counter system for Φ_k defined by $(D, Q_k, z, \mathrm{opt})$.

If, in addition, (i) Q and $\sim_{rQ,l}$ are effectively decidable, (ii) z is effectively computable, and (iii) in condition 2, $\overline{z}$ and K_l are effectively computable, then such a reduction-counter system can be effectively constructed.

This implies the existence of parallel algorithms with the stated resource bounds for the constructive versions of Max Induced d-Degree Subgraph for all $d \geq 0$, Min p-Dominating Set for all $p \geq 1$, Min Vertex Cover, Max Cut on graphs with bounded degree, and Max Leaf Spanning Tree when restricted to graphs of bounded treewidth. For a proof, see Theorem 5.3.

For the problems Min Partition into Cliques and Min Hamiltonian Path Completion we can apply Theorem 6.8 as well, but the returned solution is not exactly in the form as it would be expected (see also the proof of Theorem 5.3). Sequentially these different forms of solutions can be translated into each other in $O(n)$ time. However in parallel we know no method to do these translations in $O(\log n \log^* n)$ time with $O(n)$ operations on an EREW PRAM, or in $O(\log n)$ time with $O(n)$ operations on an EREW PRAM.

7 Additional Results and Final Comments

It is possible to generalize the results in this paper to graphs which are not necessarily connected. For this case, the definition of a special reduction system is extended.

Definition 7.1 (Special Reduction System). *Let P be a graph property, and $(\mathcal{R}, \mathcal{I})$ a reduction system for P. Let n_{max} be the maximum number of vertices in any left-hand side of a rule $r \in \mathcal{R}$. $(\mathcal{R}, \mathcal{I})$ is a* special reduction system *for P if we know positive integers n_{min} and d, $n_{min} \leq n_{max} \leq d$, such that the following conditions hold.*

1. *For each reduction rule $(H_1, H_2) \in \mathcal{R}$,*
   - *(a) if H_1 has at least one terminal, then H_1 is connected and H_1 and H_2 are open, and*
   - *(b) if H_1 is a zero-terminal graph, then $|V(H_2)| < n_{min}$.*
2. *For each graph G and each adjacency list representation of G, if $P(G)$ holds, then*
   - *(a) each component of G with at least n_{min} vertices has a d-discoverable match, and*
   - *(b) if all components of G have less than n_{min} vertices, then either $G \in \mathcal{I}$ or G contains a match which is a zero-terminal graph.*

This system can again be used in an $O(n)$ reduction algorithm. This algorithm consists of two phases: the first phase actually is algorithm Reduce, except that, instead of line 16, the algorithm checks whether each component of the current graph has at most n_{min} vertices, otherwise it returns false. In the second phase, the small components of the graph are reduced by taking components together and matching them to reduction rules. This can be done in a smart way, such that it takes $O(n)$ time, and after phase two, a graph in I remains if and only if the input graph satisfies the property. A detailed description can be found in [11].

The definitions of special constructive reduction systems and special (constructive) reduction-counter systems can be modified in the same way as the definition of special reduction systems. Furthermore we can modify algorithm Reduce-Construct and the algorithms for optimization problems in the same way as algorithm Reduce, and obtain $O(n)$ time algorithms.

Theorems 2.2, 3.2, 4.2 and 5.2 can also be shown to hold for the new type of special reduction system. For the parallel variant a similar modification can be done to the special parallel reduction system (see [11] for more details).

It is also possible to generalize the results in this paper to directed, mixed and/or labeled graphs. In the case of labeled graphs, we can allow the input graph to have a labeling of the vertices and/or edges, where the labels are taken from a set of constant size. These labels could also act as weights for finite integer index problems, e.g., we can deal with MAX WEIGHTED INDEPENDENT SET, with each vertex a weight from $\{1,2,\dots,c\}$ for some fixed c, in the same way as we dealt with MAX INDEPENDENT SET. Each of these generalizations can be handled in a very similar way as the results that are given in this paper.

For constructive decision and optimization problems, we restricted ourselves to solution domains which are vertex-edge-tuples. However, this is not always desirable. For instance, for MIN PARTITION INTO CLIQUES we would prefer to represent a solution as a partition $\{V_1,\dots,V_s\}$ of the vertices of the graph (see also the proof of Theorem 5.3). It is possible to use more general solution domains like the partition of vertices. However, these solution domains should obey a number of conditions. For instance, the function $[]$ to restrict solutions to terminal subgraphs should be defined in such a way that for each two l-terminal graphs G and H, and each $S_G \in D_{[]}(G)$, $S_H \in D_{[]}(H)$, there is at most one $S \in D(G \oplus H)$ for which $S[G] = S_G$ and $S[H] = S_H$. Furthermore, during the construction of solutions in the second phase of the reduction algorithm, it should be possible to maintain a data structure in which solutions can be adapted in $O(1)$ time.

Unfortunately, the problem of TREEWIDTH, and the related problem of PATHWIDTH are not known to have a special (parallel) constructive reduction system. Having a constructive reduction system might lead to more efficient sequential algorithms for the problem to find tree or path decompositions of bounded width (in terms of constant factors). Having a parallel constructive reduction system leads to more efficient parallel algorithms for finding a tree or path decompositions of bounded width: the gain in the amount of time is $\Theta(\log n/\log^* n)$.

An interesting problem is which graph properties have special (constructive) reduction systems. The property to have maximum degree at most some fixed constant k is an example of a property that has a special reduction system and that has yes-instances of unbounded treewidth. Also because of its associations to efficient recognition algorithms, it is interesting to know which problems have such reduction systems, and which not.

All MS-definable decision problems are of finite index, thus implying that there are efficient reduction algorithms which solve these problems (Theorem 2.2). For optimization problems this does not hold: there are MS-definable optimization problems which are not of finite integer index (Theorem 4.4), and thus these problems can not be solved with the reduction algorithms presented in Section 4. It might be interesting to find out whether there is a method with which all MS-definable optimization problems can be solved by using a type of reduction algorithm. It is also interesting to find a language like MSOL to define optimization problems which are of finite integer index. Also, one can conceive more notions similar to finite integer index, by using a different algebraic structure instead of integers and addition. It is unclear whether there exists a choice for such a structure that gives new possibilities to deal with (non-contrived) problems while keeping the same time and space bounds for the resulting algorithms.

Finally, graph reduction can also be used as a preprocessing heuristic. For instance, suppose we have a graph G on which we want to solve problem P. Now, if we have a special (constructive) reduction system for P_k, then note that all reductions from this system are also safe for P. Thus, we can use the following approach: apply reductions from the system on G, until no such reduction can be applied. Hopefully, we obtain a graph G' that is smaller than G. Now, use another approach to solve $P(G')$, be it backtracking, techniques from integer linear programming, simulated annealing, etc. Finally, translate the solution for G' back to a solution for G. The hope is that the reduction preprocessing step makes G sufficiently smaller to save time in comparison with running the algorithm to solve P directly on G.

References

[1] K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In N. Robertson and P. Seymour, editors, *Proceedings of the AMS Summer Workshop on Graph Minors, Graph Structure Theory, Contemporary Mathematics vol. 147,* pages 539–564. American Mathematical Society, 1993.

[2] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40:1134–1164, 1993.

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

[4] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.,* 7:305–314, 1986.

[5] H. L. Bodlaender. On reduction algorithms for graphs with small treewidth. In *Proceedings 19th International Workshop on Graph-Theoretic Concepts in Computer Science WG'93,* pages 45–56, 1994.

[6] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.,* 25:1305–1317, 1996.

[7] H. L. Bodlaender and B. de Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In J.-Y. Cai and C. K. Wong, editors, *Proceedings 2nd Annual International Conference on Computing and Combinatorics, COCOON'96*, pages 199–208. Springer Verlag, Lecture Notes in Computer Science, vol. 1090, 1996.

[8] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. In Z. Fülöp and F. Gécseg, editors, *Proceedings 22nd International Colloquium on Automata, Languages and Programming*, pages 268–279, Berlin, 1995. Springer-Verlag, Lecture Notes in Computer Science 944. To appear in SIAM J. Computing, 1997.

[9] 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. *Algorithmica*, 7:555–581, 1992.

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

[11] B. de Fluiter. *Algorithms for Graphs of Small Treewidth*. PhD thesis, Utrecht University, 1997.

[12] R. J. Duffin. Topology of series-parallel graphs. *J. Math. Anal. Appl.*, 10:303–318, 1965.

[13] M. R. Fellows and M. A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 520–525, 1989.

[14] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[15] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20:20–44, 1996.

[16] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 532–543. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

[17] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[18] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11:298–313, 1982.