On the Equivalence Between the Primal-Dual Schema and the Local Ratio Technique*

Reuven Bar-Yehuda† Dror Rawitz‡

February 23, 2004

Abstract

We discuss two approximation paradigms that were used to construct many approximation algorithms during the last two decades, the *primal-dual schema* and the *local ratio technique*. Recently, primal-dual algorithms were divised by first constructing a local ratio algorithm, and then transforming it into a primal-dual algorithm. This was done in the case of the 2-approximation algorithms for the *feedback vertex set* problem, and in the context of maximization algorithms. Subsequently, the nature of the connection between the two paradigms was posed as an open question by Williamson [35]. In this paper we answer this question by showing that the two paradigms are equivalent. The equivalence between the paradigms is constructive, and it implies that the integrality gap of an integer program serves as a bound to the approximation ratio when working with the local ratio technique.

Keywords: Approximation, Covering Problems, Local Ratio, Primal-Dual.

*An extended abstract containing some of these results was presented at the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'01) [10].

†Computer Science Department, Technion, Haifa 32000, Israel. E-mail: `reuven@cs.technion.ac.il`. This research was supported by the N. Haar and R. Zinn Research Fund.

‡Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel. E-mail: `rawitz@eng.tau.ac.il`. This research was done while the 2nd author was at the Technion's Department of Computer Science.

1 Introduction

1.1 Primal-Dual Schema

A key step in designing an approximation algorithm is establishing a good bound on the value of the optimum. This is where linear programming (LP) helps out. Many combinatorial optimization problems can be expressed as linear integer programs, and the value of an optimal solution to their *LP-relaxation* provides the desired bound. Clearly, the best we can hope for using this approach is to get an r-approximation algorithm, where r is the *integrality gap* of the program. One way to obtain approximate solutions is to solve the LP-relaxation and then to *round* the solution while ensuring that the cost does not change by much. Another way to go about it is to use the *dual* of the LP-relaxation in the design of approximation algorithms and their analysis. A *primal-dual* r-approximation algorithm constructs a feasible integral primal solution and a feasible dual solution such that the value of the primal solution is no more than r times (or, in the maximization case, at least $1/r$ times) the value of the dual solution. This work focuses on classical primal-dual algorithms. Specifically, those that fall within the, so called, *primal-dual schema (or method) for approximation algorithms*.

The primal-dual schema for approximation can be seen as a modified version of the *primal-dual method for solving linear programs*.[1] The primal-dual method was originally proposed by Dantzig, Ford, and Fulkerson [17]. Over the years, it became an important tool for solving combinatorial optimization problems that can be formulated as linear programs. While the *complementary slackness conditions* are imposed in the primal-dual method, we relax the dual conditions when working with the primal-dual schema. A primal-dual approximation algorithm typically constructs an approximate primal solution and a feasible dual solution simultaneously. The approximation ratio is derived from comparing the values of both solutions. The first approximation algorithm to use the primal-dual schema is Bar-Yehuda and Even's approximation algorithm for the *weighted set cover* problem [6], and since then many approximations algorithms for NP-hard optimization problems were constructed using this approach, among which are algorithms for *network design* problems (see, e.g., [33, 1, 24]). In fact, this line of research has introduced the idea of looking at *minimal* solutions (with respect to set inclusion) to the primal-dual schema.

Several primal-dual approximation frameworks were proposed in the last decade. Goemans and Williamson [24] presented a generic algorithm for a wide family of *network design* problems. They also based a subsequent survey of the primal-dual schema [25] on this algorithm. Another, more recent, survey by Williamson [35] describes the primal-dual schema and several extensions of the primal-dual approach. In [25] the authors show that the primal-dual schema can be used to explain many classical (exact and approximation) algorithms for special cases of the *hitting set* problem, such as the *shortest path*, *minimum spanning tree*, and *vertex cover* problems. Following the work of Goemans and Williamson [24], Bertsimas and Teo [12] proposed a primal-dual framework to design and analyze approximation algorithms for integer programming problems of the covering type. As in [25] this framework enforces the primal complementary slackness conditions while relaxing the dual conditions. However, in contrast to previous studies, Bertsimas and Teo [12] express each advancement step as the construction of a single valid inequality, and an increase of the corresponding dual variable (instead of an increase of several dual variables). The approximation ratio of the resulting algorithm depends upon the quality, or *strength*, of the inequalities that are used.

[1] Henceforth, we will refer to the former as the primal-dual schema and to the latter as the primal-dual method.

1.2 Local Ratio Technique

The local ratio technique uses weight subtractions. An advancement step of a local ratio algorithm typically consists of the construction of a new *weight function*, which is then subtracted from the current objective function. Each subtraction changes the optimum, but incurs a cost. The ratio between this cost and the change in the optimum is called the *effectiveness* of the weight function. The approximation ratio of a local ratio algorithm depends on the effectiveness of the weight functions it constructs.

The local ratio approach was developed by Bar-Yehuda and Even [7] in order to approximate the *set cover* and *vertex cover* problems. In this paper the authors presented a local ratio analysis to their primal-dual approximation algorithm for *set cover* [6], and a $(2 - \frac{\log\log n}{2\log n})$-approximation algorithm for vertex cover. About ten years later Bafna et al. [2] extended the *local ratio lemma* from [7] in order to construct a 2-approximation algorithm for the *feedback vertex set* problem. This algorithm was the first local ratio algorithm that used the notion of *minimal* solutions. We note that this work and the 2-approximation from [11] were essential in the design of primal-dual approximation algorithms for *feedback vertex set* [15]. Following Bafna, et al. [2], Fujito [21] presented a generic local ratio algorithm for node deletion problems with nontrivial and hereditary graph properties.[2] Later, Bar-Yehuda [4] presented a unified local ratio approach for developing and analyzing approximation algorithms for covering problems. This framework, which extends the one in [21], can be used to explain most known optimization and approximation algorithms for covering problems. Bar-Noy et al. [3] use the local-ratio technique to develop a framework for resource allocation and scheduling problems. This study was the first to present a local-ratio (or primal-dual) approximation algorithm for a natural maximization problem. A primal-dual interpretation was given in [3] as well. A detailed survey on the local ratio technique that includes recent developments such as fractional local ratio [8] can be found in [5].

1.3 Our Results

We show that the framework for covering problems by Bertsimas and Teo [12] extends the generic algorithm given in [25] by proving that every advancement step of an approximation algorithm that uses the primal-dual schema can be represented by a change in a single dual variable. Note that this was not shown in [12]. The question whether one can construct a non-dual ascend algorithm within the primal-dual schema was posed by Williamson [35]. In a way we answer this open question by showing that a non-dual ascend primal-dual algorithm can be viewed as a dual ascend algorithm. This is due to the fact that any dual change can be transformed into a change in a single dual variable.

We present two generic approximation algorithms for covering problems. The first is a recursive version of the primal-dual framework from [12], and the second is a variant of the local ratio algorithm from [4]. After presenting both frameworks we discuss the connection between them. We show that a *strong* valid inequality (in terms of [12]) and an *effective* weight function (in terms of [4]) are equivalent notions. Consequently, we prove that both frameworks for covering are one and the same. We demonstrate the combined approach on a variety of covering problems, such as *network design* problems, and the *feedback vertex set* problem. We also present a linear time approximation algorithm for the *generalized hitting set* problem (which can be viewed as the prize

[2] A graph property π is *nontrivial* if it is true for infinitely many graphs and false for infinitely many graphs; it is *hereditary* if every subgraph of a graph satisfying π also satisfies π.

collecting version of hitting set). This algorithm extends the algorithm for set cover[3] from [6] and achieves a ratio of 2 in the special case of *generalized vertex cover*. Its time complexity is significantly better than Hochbaum's [29] $O(nm \log \frac{n^2}{m})$ 2-approximation algorithm for this special case.

Next, we extend both our frameworks to include algorithms for minimization problems that are not covered by the generic algorithms from [12] and [4]. We show that the equivalence between the paradigms continues to hold (under certain conditions). We demonstrate the use of the extended frameworks on several algorithm: a 2.5-approximation algorithm for *feedback vertex set in tournaments* [14]; a 2-approximation algorithm for a non covering problem called *minimum 2-satisfiability* [27, 9]); and, a 3-approximation algorithm for a *bandwidth trading* problem [13]. We show that the equivalence continues to hold in the case of algorithms for maximization problems. We do that by developing two equivalent frameworks for maximization problems, one in each approach. Algorithms for *interval scheduling* [3] and *longest path in a DAG* are used to demonstrate our maximization frameworks.

It is important to note that the equivalence between the paradigms is constructive. That is, a primal-dual algorithm that follows our framework can be easily transformed into a local ratio algorithm, and vice versa. We also note that the nature of the connection between the two paradigms was mentioned as an open question by Williamson [35]. A corollary to this equivalence is that the *integrality gap* of a certain integer program serves as a lower bound to the approximation ratio of a local ratio algorithm.

We believe that this study contributes to the understanding of both approaches, and, especially, that it may help in the design of approximation algorithms for non covering problems, and non standard algorithms for covering problem. For example, we show that the primal-dual schema can be applied as a clean-up phase whose output is an instance of a certain type that we know how to solve by other means. This approach is quite natural in the local ratio setting, and has been used in the $(2 - \frac{\log \log n}{2 \log n})$-approximation algorithm for vertex cover [7], and the 2.5-approximation algorithm for feedback vertex set in tournaments [14].

1.4 Overview

The remainder of the paper is organized as follows. In Section 2 we define the family of problems which we consider in this paper, and state some basic facts regarding primal-dual and local ratio. In Section 3 we demonstrate the two approaches on the *Steiner tree* problem. The objective of this example is to identify the differences and similarities between the paradigms. Section 4 discusses *covering* problems. We present a generic primal-dual algorithm and a generic local ratio algorithm, both for covering problems, and we show that they are equivalent. We also show how the two generic algorithms can be applied to several covering problems. Our frameworks for minimization problems are given in Section 5. We demonstrate these frameworks by presenting several applications. Our maximization frameworks and several examples are given in Section 6.

[3]The *hitting set* problem and the *set cover* problem are equivalent problems in the sense that each is obtained from the other by switching the roles of sets and elements.

2 Preliminaries

We consider the following optimization problem: given a non negative *weight* vector $w \in \mathbb{R}^n_+$, find a solution $x \in \mathbb{N}^n$ that minimizes (or maximizes) the inner product $w \cdot x$ subject to some set $\mathcal{F}$ of feasibility constraints on x. This formulation contains, among others, all LP and IP problems. Usually, we require $x \in \{0,1\}^n$, and in this case we abuse notation by treating a vector $x \in \{0,1\}^n$ as the set of its 1 coordinates, i.e., as $\{j : x_j = 1\}$. The correct interpretation should be clear from the context.

We define the following for a minimization (maximization) problem $(\mathcal{F}, w)$. A vector x is called a *feasible solution* if x satisfies the constraints in $\mathcal{F}$. A feasible solution x^* is *optimal* if every feasible solution x satisfies $w \cdot x^* \leq w \cdot x$ ($w \cdot x^* \geq w \cdot x$). We denote by Opt the value of an optimal solution, i.e., the optimum value. A feasible solution x is called an *r-approximation* or *r-approximate* if $w \cdot x \leq r \cdot w \cdot x^*$ ($w \cdot x \geq \frac{1}{r} \cdot w \cdot x^*$), where x^* is an optimal solution. An algorithm is called an *r-approximation algorithm* if it returns r-approximate solutions. Namely, an r-approximation algorithm returns a feasible solution whose weight is no more than r (at least $1/r$) times the optimum weight.

2.1 Primal-Dual

This section is written in terms of minimization problems. Similar arguments can be given in the maximization case. Also, in the sequel we assume basic knowledge of linear programming. (See, e.g., [32, 31] for more details about linear programming.)

Consider the following linear program,

$$\begin{array}{lll} \min & \sum_{j=1}^n w_j x_j & \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j \geq b_i & \forall i \in \{1,\ldots,m\} \\ & x_j \geq 0 & \forall j \in \{1,\ldots,n\} \end{array}$$

and its dual,

$$\begin{array}{lll} \max & \sum_{i=1}^n b_i y_i & \\ \text{s.t.} & \sum_{i=1}^n a_{ij} y_i \leq w_j & \forall j \in \{1,\ldots,n\} \\ & y_i \geq 0 & \forall i \in \{1,\ldots,m\} \end{array}$$

A primal-dual r-approximation algorithm for a minimization problem produces an integral primal solution x and a dual solution y such that the weight of the primal solution is no more than r times the value of dual solution. Namely, it produces an integral solution x and a solution y such that

$$wx \leq r \cdot by \tag{1}$$

The Weak Duality Theorem implies that x is r-approximate.

One way to design an algorithm that finds a pair of primal and dual solutions that satisfy (1) is to restrict our attention to a specific kind of pairs of primal and dual solutions. Consider a primal solution x and a dual solution y. The Duality Theorem provides us with a way to characterize a pair of optimal solutions. Specifically, x and y are optimal if and only if the following conditions, called the *complementary slackness conditions*, are satisfied:

$$\begin{array}{llll} \text{Primal conditions:} & \forall j,\ x_j > 0 & \Rightarrow & \sum_{i=1}^m a_{ij} y_i = w_j \\ \text{Dual conditions:} & \forall i,\ y_i > 0 & \Rightarrow & \sum_{j=1}^n a_{ij} x_j = b_i \end{array}$$

However, we are interested in approximate solutions, thus it seems natural to relax the complementary slackness conditions. Consider an integral primal solution x and a dual solution y that satisfy the following conditions, called the *relaxed complementary slackness conditions* [34]:

$$\begin{array}{lll} \text{Relaxed Primal:} & \forall j,\ x_j > 0 \ \Rightarrow & w_j/r_1 \leq \sum_{i=1}^m a_{ij} y_i \leq w_j \\ \text{Relaxed Dual:} & \forall i,\ y_i > 0 \ \Rightarrow & b_i \leq \sum_{j=1}^n a_{ij} x_j \leq r_2 \cdot b_i \end{array}$$

Then,

$$\sum_{j=1}^n w_j x_j \leq \sum_{j=1}^n r_1 \cdot \left(\sum_{i=1}^m a_{ij} y_i \right) x_j = r_1 \cdot \sum_{i=1}^m \left(\sum_{j=1}^n a_{ij} x_j \right) y_i \leq r_1 \cdot r_2 \cdot \sum_{i=1}^m b_i y_i$$

which means that x is $r_1 \cdot r_2$-approximate.

In this study we consider algorithms in which $r_1 = 1$, that is, algorithms that only relax the dual complementary slackness conditions. (Algorithms that relax the primal conditions are studied in [20].) Typically, such an algorithm constructs an integral primal solution x and a feasible dual solution y simultaneously. It starts with an infeasible primal solution and a feasible dual solution (usually, $x = 0$ and $y = 0$). It iteratively raises the dual solution, and improves the feasibility of the primal solution. In each iteration the dual solution is increased while ensuring that the relaxed dual conditions are satisfied. Also, a primal variable can be increased only if its corresponding primal condition is obeyed.

2.2 Local Ratio

Say we want to construct an r-approximation algorithm for a minimization problem. A key step in the design of such an algorithm is to establish a good lower bound b on the weight of the optimal solution. This bound can later be used in the analysis to prove that the solution found by the algorithm is r-approximate by showing that its weight is no more than $r \cdot b$. The local ratio technique uses a "local" variation of this idea. In essence, the idea is to break down the weight w of the solution found by the algorithm into a sum of "partial weights" $w = w_1 + w_2 + \ldots + w_k$, and similarly break down the lower bound b into $b = b_1 + b_2 + \ldots + b_k$, and to show that $w_i \leq r \cdot b_i$ for all i. The breakdown of w and b is determined by the manner in which the solution is constructed by the algorithm. In fact, the algorithm constructs the solution in such a manner as to ensure that such a breakdown exists. Put differently, at the ith step, the algorithm "pays" $r \cdot b_i$ and manipulates the problem instance so that the optimum drops by at least b_i.

The local-ratio technique is based on the following Theorem. (The proof is given for purposes of completeness.)

Theorem 1 (Local Ratio Theorem [3]) *Let $(\mathcal{F}, w)$ be a minimization (maximization) problem, and let w, w_1, and w_2 be weight functions such that $w = w_1 + w_2$. Then, if x is r-approximate with respect to $(\mathcal{F}, w_1)$ and with respect to $(\mathcal{F}, w_2)$, then x is r-approximate with respect to $(\mathcal{F}, w)$.*

Proof. Let x^*, x_1^*, x_2^* be optimal solutions with respect to $(\mathcal{F}, w), (\mathcal{F}, w_1)$, and $(\mathcal{F}, w_2)$ respectively. Then, in the minimization case we have,

$$wx = w_1 x + w_2 x \leq r \cdot w_1 x_1^* + r \cdot w_2 x_2^* \leq r \cdot (w_1 x^* + w_2 x^*) = r \cdot wx^* \ .$$

For the maximization case simply replace $\leq$ by $\geq$ and r by $\frac{1}{r}$. ■

Note that $\mathcal{F}$ can include arbitrary feasibility constraints, and not just linear, or linear integer, constraints. Nevertheless, all successful applications of the local ratio technique to date involve problems in which the constraints are linear.

Usually, the Local Ratio Theorem is used in the following manner. Given a problem instance with a non negative weight function w, we find a non negative weight function $\delta \le w$ such that every *minimal* solution (with respect to set inclusion) is r-approximate with respect to δ. Then, we recursively find a minimal solution that is r-approximate with respect to $w - \delta$. By the Local Ratio Theorem this solution is r-approximate with respect to the original weights w. The recursion terminates when a minimal r-approximate solution can be found directly, which usually occurs when the problem instance is an empty instance, or when the weights have evolved to the point that the set of all zero-weight elements constitutes a feasible (and hence optimal) solution. Note that the scheme just described is tail recursive and can thus be implemented iteratively rather than recursively.

3 An Introductory Example: The Steiner Tree Problem

In this section we compare two approximation algorithms for the *Steiner tree* problem, one based on the primal-dual schema and the other on the local ratio technique. The two algorithms are not new, but they demonstrate how one usually uses both paradigms, and thus help us to identify differences and similarities between the two approaches. Also, this example will be useful in the next section. We start with the definition of the problem.

Given a graph $G = (V, E)$, and a non empty set of *terminals* $T \subseteq V$, a *Steiner tree* is a subtree of G that connects all the vertices in T. Given a non negative weight function w on the edges, the Steiner tree problem is to find a minimum weight Steiner tree, where the weight of a tree is the total weight of its edges. (We consider trees to be sets of edges.)

We are interested in Steiner trees that are *minimal* with respect to set inclusion. Namely, a Steiner tree F is minimal if $F \setminus \{e\}$ is not a Steiner tree for every edge $e \in F$. Observe that a Steiner tree is minimal if and only if every leaf in the tree is a terminal. For an edge $e \in E$ we denote the number of terminals incident to e, or the *terminal degree* of e, by $\tau(e)$, i.e., $\tau(e) = |e \cap T|$.

Lemma 1 *Let F be a minimal Steiner tree. Then,* $|T| \le \sum_{e \in F} \tau(e) \le 2 \cdot |T| - 2$.

Proof Sketch. The first inequality follows from the fact that every terminal must be incident to some edge in F. The second inequality can be proven as follows. First, we pick an arbitrary terminal r to be the root of the Steiner tree. Next, we place a total of $2 \cdot |T| - 2$ coins on the terminals—two coins on each terminal in $T \setminus \{r\}$, and show that we can reassign the coins such that there will be at least $\tau(e)$ coins on each edge $e \in F$. Consider a terminal $t \in T \setminus \{r\}$, and let u be the parent of t. Also, let s be the terminal which is closest to u on the path from u to r, and let v be s's child on that path. t places one coin on the edge (t, u), and another coin on the edge (v, s). (If $u = s$ and $v = t$ then two coins are placed on (t, u).) It is not hard to show that, because the leaves of F are terminals, at least $\tau(e)$ coins are placed on every edge $e \in F$. ■

(See [25, 5] for a slightly different proof of a more general claim.)

3.1 Primal-Dual

A typical first step in the design of a primal-dual approximation algorithm is to find a suitable formulation of the problem at hand as a linear integer program. Indeed, we start with such a formulation of the Steiner tree problem. We say that a subset $S \subseteq V$ *splits* T if $\emptyset \subsetneq S \cap T \subsetneq T$. Let $\textsc{Split}(T)$ be the set of all subsets of V that split T, i.e., $\textsc{Split}(T) = \{S : \emptyset \subsetneq S \cap T \subsetneq T\}$. The Steiner tree problem can be formulated by the following linear integer program:

$$
\begin{array}{lll}
\text{(ST)} \quad \min & \displaystyle\sum_{e \in E} w(e) x_e & \\
\text{s.t.} & \displaystyle\sum_{e \in (S,\bar{S})} x_e \geq 1 & \forall S \in \textsc{Split}(T) \\
& x_e \in \{0,1\} & \forall e \in E
\end{array}
$$

where $(S, \bar{S})$ denotes the set of edges having exactly one endpoint in S. We get an LP-relaxation by replacing the last set of constraints by: $x_e \geq 0$, $\forall e \in E$. The corresponding dual program is:

$$
\begin{array}{lll}
\max & \displaystyle\sum_{S \in \textsc{Split}(T)} y_S & \\
\text{s.t.} & \displaystyle\sum_{S : e \in (S,\bar{S})} y_S \leq w(e) & \forall e \in E \\
& y_S \geq 0 & \forall S \in \textsc{Split}(T)
\end{array}
$$

Algorithm **PD-ST** is a primal-dual approximation algorithm for the Steiner tree problem. It is a specific implementation of the generic algorithm given in [24]. The algorithm starts with $|V|$ components—each containing a single vertex. These components are induced by the solution F. In the ℓth iteration it raises the dual variables that correspond to components that split T until some dual constraint becomes tight. Then, an edge that corresponds to some tight dual constraint is added to F, and the components are updated accordingly. This process terminates when all terminals are in the same component. Then, F is turned into a minimal Steiner tree using reverse deletion.

Algorithm PD-ST(G, w)

$F \leftarrow \emptyset$
$y \leftarrow 0$
$\mathcal{C}_0 \leftarrow \{\{v\} : v \in V\}$
$\ell \leftarrow 0$
While $\exists C \in \mathcal{C}_\ell$ such that C splits T
    $\ell \leftarrow \ell + 1$
    Increase y_C uniformly for every $C \in \mathcal{C}$ that splits T
    until some dual constraint becomes tight
    Let $e_\ell = (u, v)$, such that $u \in C_i$ and $v \in C_j$,
    be an edge that corresponds to a tight dual constraint
    $F \leftarrow F \cup \{e_\ell\}$
    $\mathcal{C}_\ell \leftarrow \mathcal{C}_{\ell-1} \cup \{C_i \cup C_j\} \setminus \{C_i, C_j\}$
For $j \leftarrow \ell$ down-to 1
    If $F \setminus \{e_j\}$ is feasible then $F \leftarrow F \setminus \{e_j\}$
Output F

First, we show that Algorithm **PD-ST** produces feasible solutions. Consider a solution F returned by the algorithm. Observe that all the terminals are in the same component, otherwise the algorithm would not have terminated. Also, due to lines 11–12 F is a minimal Steiner tree.

We need only prove that Algorithm **PD-ST** produces 2-approximate solutions. Let y be the dual solution corresponding to a solution F that was output by the algorithm. By the Weak Duality Theorem $\sum_{S \in \text{SPLIT}(T)} y_S \le \text{OPT}$. Thus, in order to show that F is 2-approximate, it is enough to prove that $\sum_{e \in F} w(e) \le 2 \cdot \sum_{S \in \text{SPLIT}(T)} y_S$.

In the ℓth iteration the algorithm raises y_C for every component C that splits T, therefore

$$\sum_{S \in \text{SPLIT}(T)} y_S = \sum_{\ell=1}^{t} \epsilon_\ell \left|\mathcal{C}'_\ell\right|$$

where ϵ_ℓ is the dual increase at the ℓth iteration, and $\mathcal{C}'_\ell \subseteq \mathcal{C}_\ell$ is the set of components that split T (*active* components in the terminology of [24]). On the other hand, only edges that correspond to tight dual constraints are taken into the solution F, hence

$$\sum_{e \in F} w(e) = \sum_{e \in F} \sum_{S : e \in (S, \bar{S})} y_S = \sum_{e \in F} \sum_{S : e \in (S, \bar{S})} \sum_{\ell : S \in \mathcal{C}'_\ell} \epsilon_\ell = \sum_{\ell=1}^{t} \epsilon_\ell \sum_{C \in \mathcal{C}'_\ell} \left|(C, \bar{C}) \cap F\right| .$$

Thus, it is enough to prove that for every $\ell \in \{1, \ldots, t\}$,

$$\sum_{C \in \mathcal{C}'_\ell} \left|(C, \bar{C}) \cap F\right| \le 2 \cdot \left|\mathcal{C}'_\ell\right| .$$

Observe that for a component $C \in \mathcal{C}'_\ell$, $\left|(C, \bar{C}) \cap F\right|$ is the number of edges in F with one endpoint in C. If we could prove that $\left|(C, \bar{C}) \cap F\right| \le 2$ for every $C \in \mathcal{C}'_\ell$, then we are done. However, this is not necessarily true. Instead, we prove an "amortized" version of this claim. That is, we prove that the average number of edges in F with one endpoint in a component $C \in \mathcal{C}'_\ell$ is no more that two. We remark that by doing that we actually prove that the relaxed dual complementary slackness conditions are satisfied (as shown in the next chapter).

Consider the ℓth iteration, and define a multi-graph (a graph that may contain multiple edges between pairs of vertices) $G^\ell = (V^\ell, E^\ell)$ as follows. Each vertex in V^ℓ corresponds to a component $C \in \mathcal{C}_\ell$. We refer to a vertex u as a terminal in G^ℓ if the corresponding component in G contains at least one terminal (i.e., if the corresponding component is in $\mathcal{C}'_\ell$). We denote the set of terminals in G^ℓ by T^ℓ. Let u, v be vertices in G^ℓ and let C_u, C_v be the corresponding components. E^ℓ contains a copy of the edge (u, v) for every edge $(x, y) \in E$ such that $x \in C_u, y \in C_v$, and the weight of this copy is $w(x, y)$. Consider the set of edges F^ℓ that is induced by F in G^ℓ. Clearly,

$$\sum_{C \in \mathcal{C}'_\ell} \left|(C, \bar{C}) \cap F\right| = \sum_{v \in T^\ell} \left|E^\ell(v) \cap F^\ell\right| = \sum_{e \in F^\ell} \tau_{G^\ell}(e)$$

where $E^\ell(v)$ is the set of edges incident on v (in G^ℓ). We claim that F^ℓ is a minimal Steiner tree in G^ℓ. To see this observe that in the ℓth iteration the terminals in each component C are connected in G (by edges within each component). Moreover, due to the reverse deletion phase (Lines 11-12) the edges in F^ℓ form a minimal Steiner tree in G^ℓ. Thus, by Lemma 1, we know that

$$\sum_{e \in F^\ell} \tau_{G^\ell}(e) \le 2 \cdot |T^\ell| - 2 = 2 \cdot |C'_\ell| - 2$$

and we are done.

3.2 Local Ratio

The following local ratio approximation algorithm appeared in [4] (though in less detail). In the course of its execution, the algorithm modifies the graph by performing *edge contractions*. Contracting an edge (u, v) consists of "fusing" its two endpoints u and v into a single (new) vertex z. The edge connecting u and v is deleted and every other edge incident on u or v becomes incident on z instead. In addition, if either u or v are terminals then z is a terminal too.

Algorithm LR-ST(G, T, w)

1. If G contains only one terminal then return $\emptyset$
2. Else:
3. Let $\epsilon = \min_e \{w(e)/\tau(e)\}$
4. Define the weight function $\delta(e) = \epsilon \cdot \tau(e)$
5. Let e be an edge such that $w(e) = \delta(e)$
6. Let (G', T') be the instance obtained by contracting e
7. $F' \leftarrow$ **LR-ST**$(G', T', w - \delta)$
8. If F' is not feasible then return $F = F' \cup \{e\}$
9. Else, return $F = F'$

Note the slight abuse of notation in Line 7. The weight function in the recursive call is not $w - \delta$ itself, but rather the restriction on G'. We will continue to silently abuse notation in this manner.

We show that Algorithm **LR-ST** returns a minimal Steiner tree. The proof is by induction on the number of terminals. At the recursion basis the solution returned is the empty set, which is both feasible and minimal. For the inductive step, by the inductive hypothesis F' is a minimal Steiner tree with respect to G' and T'. Also, we add e to F only if we have to. Therefore, F is a minimal Steiner tree with respect to G and T.

It remains to prove that Algorithm **LR-ST** produces 2-approximate solutions. The proof is also by induction on the number of terminals. In the base case the solution returned is the empty set, which is optimal. For the inductive step, by the inductive hypothesis F' is 2-approximate with respect to G', T' and $w - \delta$. Since $(w - \delta)(e) = 0$, the weight of F with respect to $w - \delta$ equals to that of F', and the optimum value for (G, T) with respect to $w - \delta$ cannot be smaller than the optimum value for (G', T') because if F^* is an optimal solution for (G, T) then $F^* \setminus \{e\}$ is a feasible solution of the same weight for (G', T'). Thus, F is 2-approximate with respect to $(G, T, w - \delta)$. By Lemma 1, any minimal Steiner tree in G is 2-approximate with respect to δ. Thus, by the Local Ratio Theorem, F is 2-approximate with respect to (G, T, w) as well.

3.3 Primal-Dual vs. Local Ratio

Algorithm **PD-ST** and Algorithm **LR-ST** represent many algorithms in the literature in the sense that each of them can be viewed as a standard use of the corresponding paradigm. Algorithm **PD-ST** heavily relies on LP-duality. It is based on a predetermined linear program and its dual program, and its analysis is based on the comparison between the values of an integral primal solution and a dual solution. Algorithm **PD-ST** is iterative, and in each iteration the dual solution is changed. In a sense, the dual solution can be viewed as the book-keeper of the algorithm. On the other hand, Algorithm **LR-ST** does not use linear programming. Instead, it relies upon weight decompositions, and a Local Ratio Theorem. As in this case, local ratio algorithms are typically recursive, and in each recursive call the weights are decomposed and the instance is modified. The decomposition is determined by a weight function defined in the current recursive call. Thus, at least at first glance, the two algorithms and their analyses seem very different.

Having said all that, we turn to the similarities between the algorithms. Both algorithms use the same combinatorial property (Lemma 1) to achieve an approximate solution. The performance ratio of both algorithms was proven locally. That is, it was shown, using the above mentioned property, that in each iteration/decomposition a certain ratio holds. Also, both solutions use a reverse deletion phase. In the next section we show that this is no coincidence. The equivalence between the paradigms is based on the fact that "good" valid inequalities are equivalent to "good" weight functions. We shall also see that the changes in the dual during a primal-dual algorithm are strongly connected to the values of ϵ that are chosen in the recursive calls of a local ratio algorithm.

4 Covering Problems

Perhaps the most famous covering problem is the *set cover* problem. In this problem we are given a collection of sets $\mathcal{C} = \{S_1, \ldots, S_m\}$, and a weight function w on the sets. The objective is to find a minimum-weight collection of sets that "covers" all elements. In other words, a collection $\mathcal{C}' \subseteq \mathcal{C}$ is a *set cover* if each element in $\bigcup_{i=1}^{m} S_i$ is contained in some set from $\mathcal{C}'$, and we aim to find a set cover of minimum weight. Consider a set cover $\mathcal{C}'$. Clearly, if we add sets from $\mathcal{C} \setminus \mathcal{C}'$ to $\mathcal{C}'$ the resulting collection is also a set cover. This property is shared by all *covering problems*. A minimization problem $(\mathcal{F}, w)$ is called a covering problem if (1) $x \in \{0,1\}^n$; and (2) any extension of a feasible solution to any possible instance is always feasible. In this case, we call the set of constraints $\mathcal{F}$ *monotone*. Note that a monotone set of linear constraints typically contains inequalities with non negative coefficients.

The family of covering problems contains a broad range of optimization problems. Many of them, such as *vertex cover*, *feedback vertex set*, and *Steiner tree* were studied extensively during the last two decades. In fact, both the primal-dual schema and the local ratio technique were developed for the purpose of finding good approximate solutions for the *set cover* problem, and its special case, the *vertex cover* problem.

Primal-dual approximation algorithms for covering problems traditionally reduce the size of the instance at hand in each iteration by adding an element $j \in \{1, \ldots, n\}$ whose corresponding dual constraint is tight to the primal solution (see, e.g., [25, 12]). Local ratio algorithms for covering problems implicitly add all zero weight elements to the solution, and, therefore, reduce the size of the instance in each step as well (see, e.g., [4]). In order to implement this we alter the problem definition by adding a set (or vector), denoted by z, which includes elements that are considered (at least, temporarily) to be taken into the solution. This makes it easier to present primal-dual algorithms recursively, and to present local ratio algorithms in which the addition of zero weight elements to the partial solution is explicit.

More formally, given a monotone set of constraints $\mathcal{F}$, a weight function w, and a vector $z \in \{0,1\}^n$, we are interested in the following problem. Find a vector $x \in \{0,1\}^n$ such that (1) $z \cap x = \emptyset$; (2) $x \cup z$ satisfies $\mathcal{F}$; And, (3) minimizes the inner product $w \cdot x$. (When $z = \emptyset$ we get the original problem $(\mathcal{F}, w)$.) z can be viewed as an additional monotone constraint, and therefore this problem is a covering problem. The definitions of a feasible solution, an optimal solution, and an r-approximate solution can be understood in a straightforward manner. We denote the set of feasible solutions with respect to $\mathcal{F}$ and z by $\textsc{Sol}(\mathcal{F}, z)$. Also, a feasible solution x is called *minimal* (with respect to set inclusion) if for all $j \in x$ the vector $z \cup x \setminus \{j\}$ is not feasible.

We remark that the use of this terminology is very useful in the context of this paper, i.e., for presenting generic algorithms, and for showing the equivalence between the two paradigms. However, it may be inept for constructing an approximation algorithm for a specific problem.

4.1 A Primal-Dual Framework for Covering Problems

Many primal-dual algorithms in the literature follow the outline of the generic algorithm given by Goemans and Williamson [25]. We show that the framework for covering problems by Bertsimas and Teo [12] extends the generic algorithm from [25]. Afterwards, we present our own recursive primal-dual framework for approximating covering problems that is based on the one from [12].

Goemans and Williamson base their generic algorithm on the *hitting set* problem, which is defined as follows: Given subsets $T_1, \ldots, T_q$ of a ground set E and a non negative cost w_e for every element $e \in E$, find a minimum-cost subset $x \subseteq E$ such that $x \cap T_i \neq \emptyset$ for every $i \in \{1, \ldots, q\}$. In turns out that many known problems (shortest path, vertex cover, etc.) are special cases of the hitting set problem. The hitting set problem can be formulated as follows:

$$\begin{array}{lll} \min & \sum_{e \in E} w_e x_e & \\ \text{s.t.} & \sum_{e \in T_i} x_e \geq 1 & \forall i \in \{1, \ldots, q\} \\ & x_e \in \{0, 1\} & \forall e \in E \end{array}$$

where $x_e = 1$ if and only if $e \in x$. The LP-relaxation and the corresponding dual program are:

$$\begin{array}{lll} \min & \sum_{e \in E} w_e x_e & \\ \text{s.t.} & \sum_{e \in T_i} x_e \geq 1 & \forall i \in \{1, \ldots, q\} \\ & x_e \geq 0 & \forall e \in E \end{array} \qquad \begin{array}{lll} \max & \sum_{i=1}^{q} y_i & \\ \text{s.t.} & \sum_{i: e \in T_i} y_i \leq w_e & \forall e \in E \\ & y_i \geq 0 & \forall i \in \{1, \ldots, q\} \end{array}$$

The primal complementary slackness conditions are:

$$e \in x \implies \sum_{i: e \in T_i} y_i = w_e \ ,$$

and the dual complementary slackness conditions are:

$$y_i > 0 \implies |x \cap T_i| = 1 \ .$$

Goemans and Williamson's algorithm that is given below (and is taken from [25, Page 158]) starts with the feasible dual solution $y = 0$ and the non feasible primal solution $x = \emptyset$. It iteratively increases the primal and dual solutions until the primal solution becomes feasible. In each iteration, if x is not feasible then there exists a set T_k such that $x \cap T_k = \emptyset$. Such a subset is called *violated*. Indeed, the increase of the dual solution involves some dual variables corresponding to violated sets. Specifically, the increase of the dual variables depends on a *violation oracle* (called $\textsc{Violation}$). In each iteration the violation oracle supplies a collection of violated subsets $\mathcal{V} \subseteq \{T_1, \ldots, T_q\}$[4], and the dual variables that correspond to subsets in $\mathcal{V}$ are increased *simultaneously and at the same speed*. When x becomes feasible a *reverse delete step* is performed. This step discards as many elements as possible from the primal solution x as long as feasibility is maintained

[4]Some subsets in $\mathcal{V}$ may not be violated. See [25] for more details.

Algorithm GW

1. $y \leftarrow 0$
2. $x \leftarrow \emptyset$
3. $j \leftarrow 0$
4. While x is not feasible
5. $j \leftarrow j + 1$
6. $\mathcal{V} \leftarrow$ Violation(x)
7.   Increase y_k uniformly for all $T_k \in \mathcal{V}$ until
       $\exists e_j \notin x : \sum_{i: e_\ell \in T_i} y_i = w_{e_j}$
8. $x \leftarrow x \cup \{e_j\}$
9. $\ell \leftarrow j$
10. For $j \leftarrow \ell$ down-to 1
11.   If $x \setminus \{e_j\}$ is feasible then $x \leftarrow x \setminus \{e_j\}$
12. Output x

Let x^f be the set output by the algorithm, and let ϵ_j denote the increase of the dual variables corresponding to $\mathcal{V}_j$ in iteration j. Thus, $y_i = \sum_{j: T_i \in \mathcal{V}_j} \epsilon_j$, $\sum_{i=1}^{q} y_i = \sum_{j=1}^{\ell} |\mathcal{V}_j| \, \epsilon_j$, and

$$\begin{aligned} w(x^f) &= \sum_{e \in x^f} w_e = \sum_{e \in x^f} \sum_{i: e \in T_i} y_i = \sum_{i=1}^{q} \left| x^f \cap T_i \right| y_i \\ &= \sum_{i=1}^{q} \left| x^f \cap T_i \right| \sum_{j: T_i \in \mathcal{V}_j} \epsilon_j = \sum_{j=1}^{\ell} \left(\sum_{T_i \in \mathcal{V}_j} \left| x^f \cap T_i \right| \right) \epsilon_j \end{aligned}$$

From these expressions it is clear that the cost of x^f is at most the value of the dual solution times r (and, therefore, x^f is r-approximate) if for all $j \in \{1, \ldots, \ell\}$

$$\sum_{T_i \in \mathcal{V}_j} \left| x^f \cap T_i \right| \leq r \cdot |\mathcal{V}_j| \tag{2}$$

Examine iteration j of the reverse deletion step. We know that when e_j was considered for removal, no element $e_{j'}$ with $j' < j$ has been already removed. Thus, after e_j is considered for removal the temporary solution is $x^j = x^f \cup \{e_1, \ldots, e_{j-1}\}$. Observe that x^j is feasible and $x^j \setminus \{e\}$ is not feasible for all $e \in x^j \setminus \{e_1, \ldots, e_{j-1}\}$. x^j is called a *minimal augmentation* of $\{e_1, \ldots, e_{j-1}\}$ in [25]. Moreover,

$$\sum_{T_i \in \mathcal{V}_j} \left| x^f \cap T_i \right| \leq \sum_{T_i \in \mathcal{V}_j} \left| x^j \cap T_i \right| \ .$$

Therefore, to achieve the bound given in (2) Goemans and Williamson [25] have set the following requirement on every collection of subsets $\mathcal{V}_j$:

$$\sum_{T_i \in \mathcal{V}_j} \left| x' \cap T_i \right| \leq r \cdot |\mathcal{V}_j|$$

for any minimal augmentation x' of $\{e_1, \ldots, e_{j-1}\}$.

To summarize, in order to construct an r-approximate solution, in each iteration of the algorithm, we seek a collection $\mathcal{V}$ such that $\sum_{T_i \in \mathcal{V}} |x' \cap T_i| \leq r \cdot |\mathcal{V}|$ for any minimal augmentation x' of the current (non feasible) primal solution x. In essence we seek a collection $\mathcal{V}$ that satisfies a sort of amortized relaxed version of the dual complementary slackness conditions. We now formalize this demand from a collection of violated subsets in our terminology. (Note that we have replaced x with z and x' with x.)

Definition 1 *A collection $\mathcal{V} \subseteq \{T_1, \ldots, T_q\}$ is called* r-effective *with respect to $(\mathcal{F}, w, z)$, if $\sum_{T_i \in \mathcal{V}} |x \cap T_i| \leq r \cdot |\mathcal{V}|$ for any minimal feasible solution x with respect to $(\mathcal{F}, z)$.*

As did Bertsimas and Teo [12] we prefer to speak in terms of inequalities. An inequality is referred to as *valid* if any feasible solution to the problem at hand satisfies this inequality. For example, given an IP formulation of a problem, any inequality that appears in this formulation is valid. The following definition uses terms of inequalities and extends the previous definition.

Definition 2 *A set of valid inequalities $\{\alpha^1 x \geq \beta^1, \ldots, \alpha^k x \geq \beta^k\}$ is called* r-effective *with respect to $(\mathcal{F}, w, z)$, if $\alpha^k_j = 0$ for every k and $j \in z$, and any integral minimal feasible solution x with respect to $(\mathcal{F}, z)$ satisfies: $\sum_{i=1}^k \alpha^i x \leq r \cdot \sum_{i=1}^k \beta^i$. If this is true for* any integral feasible solution *the set is called* fully r-effective. *If an r-effective set contains a single inequality, we refer to this inequality as* r-effective.

We remark that we require $\alpha^k_j = 0$ for every k and every $j \in z$ since in general we discuss inequalities with respect to $(\mathcal{F}, z)$ and not with respect to $\mathcal{F}$. If $z = \emptyset$ we sometimes say that the set (or the inequality) is r-effective with respect to $\mathcal{F}$.

An r-effective collection $\mathcal{V}$ can be understood as the following r-effective collection of valid inequalities: $\{\sum_{e \in T_i} x_e \geq 1 : T_i \in \mathcal{V}\}$. However, Definition 1 allows the use of other kinds of inequalities, and therefore extends Definition 2. Thus, it would seem that our goal is to find an r-effective set of valid inequalities in each iteration. However, we show that it is enough to construct a *single* r-effective valid inequality for that purpose. Consider an r-effective set of valid inequalities $S = \{\alpha^1 x \geq \beta^1, \ldots, \alpha^k x \geq \beta^k\}$, and examine the inequality

$$\sum_{i=1}^{k} \alpha^i x = \sum_{j=1}^{n} (\sum_{i=1}^{k} \alpha^i)_j x_j \geq \sum_{i=1}^{k} \beta^i$$

that we get by summing up the inequalities in S. Since S is r-effective we know that $\sum_{i=1}^k \alpha^i x \leq r \cdot \sum_{i=1}^k \beta^i$, and we have found our r-effective inequality. Thus, our goal, in each iteration of the algorithm, is to find an inequality $\alpha x \geq \beta$ such that any minimal solution satisfies the following relaxed dual condition:

$$y_i > 0 \implies \alpha \cdot x \leq r\beta \ .$$

As an example, examine the 2-approximation algorithm for the Steiner tree problem (Algorithm **PD-ST** of Section 3). The r-effective collection of sets that is chosen by the algorithm in the ℓth iteration is $\mathcal{V} = \{(C, \bar{C}) : C \in \mathcal{C}'_\ell\}$. The corresponding r-effective collection of valid inequalities is $S = \{\sum_{e \in (C, \bar{C})} x_e \geq 1 : C \in \mathcal{C}'_\ell\}$. Consider the inequality that we get by summing up the inequalities in S:

$$\sum_{C \in \mathcal{C}'_\ell} \sum_{e \in (C, \bar{C})} x_e = \sum_{e \in E^\ell} \tau_{G^\ell}(e) x_e \geq |\mathcal{C}'_\ell| \tag{3}$$

where E^ℓ is the edge set of G^ℓ. Clearly, Inequality 3 is valid, and, by Lemma 1 it is also 2-effective. Notice that the coefficients of Inequality 3 and the weights that are used in Algorithm **LR-ST** are identical. As we shall see in the sequel that this is no coincidence.

Bertsimas and Teo [12] proposed a generic algorithm to design and analyze primal-dual approximation algorithms for problems of the following type:

$$\begin{array}{ll} \min & wx \\ \text{s.t.} & Ax \geq b \\ & x \in \{0,1\}^n \end{array}$$

where all coefficients, A, b and w, are nonnegative. This algorithm utilizes a single valid inequality (or dual variable) in each iteration, and uses it to modify the current instance. After this modification the primal solution is augmented, and this makes it possible to reduce the size of the problem in each iteration. Thus, the algorithm terminates after no more than n iterations. The performance of this algorithm depends on the choice of the inequalities. In fact, it corresponds to what Bertsimas and Teo call the *strength* of an inequality, which, in our terminology, is the minimal value of r for which it is r-effective. It is important to note that, unlike other primal-dual algorithms, this algorithm constructs new valid inequalities during its execution. Another difference is that it uses the weight vector in order to measure the tightness of the dual constraints. Thus, in each iteration it decreases the weights according to the inequality that was used. In fact, this study was inspired by the similarity between this weight decrease and its local ratio counterpart.

Algorithm **PDcov** is a recursive version of the algorithm from [12]. The initial call is **PDcov**$(\emptyset, w, 1)$. (Note that the third parameter is used only for purposes of analysis.) Informally, it can be viewed as follows: construct an r-effective inequality; update the corresponding dual variable and w such that w remains non negative; find an element j whose weight is zero; add j to the temporary partial solution z; then recursively solve the problem with respect to $\mathcal{F}, z$ and the new weights (the termination condition of the recursion is met when the empty set becomes feasible); finally, j is added to the solution x only if it is necessary.

Algorithm PDcov(z, w, k)

1. If $\emptyset \in \textsc{Sol}(\mathcal{F}, z)$ return $\emptyset$
2. Construct a valid inequality $\alpha^k x \geq \beta^k$ which is r-effective w.r.t. $(\mathcal{F}, z)$
3. $y_k \leftarrow \max\left\{\epsilon \,:\, w - \epsilon\alpha^k \geq 0\right\}$
4. Let $j \notin z$ be an index for which $w_j = y_k \alpha^k_j$
5. $x \leftarrow$ **PDcov**$(z \cup \{j\}, w - y_k\alpha^k, k+1)$
6. If $x \notin \textsc{Sol}(\mathcal{F}, z)$ then $x \leftarrow x \cup \{j\}$
7. Return x

The following analysis is based on the corresponding analysis from [12].

We start by proving that Algorithm **PDcov** returns minimal feasible solutions with respect to (F, z). We prove this by induction on the recursion. At the recursion basis the solution returned is the empty set, which is both feasible and minimal. For the inductive step, let x' be the solution returned by the recursive call in Line 5. x' is feasible with respect to $(\mathcal{F}, z \cup \{j\})$ by the inductive hypothesis, therefore x is feasible with respect to (F, z). We show that $x \setminus \{i\}$ is not feasible for every $i \in x$. For the case where $i \neq j$, if $x \setminus \{i\}$ is feasible with respect to $(\mathcal{F}, z)$ then $x' \setminus \{i\}$ is feasible with respect to $(\mathcal{F}, z \cup \{j\})$ in contradiction with the minimality of x'. The case where $i = j$, which is relevant only when $x = x' \cup \{j\}$, is trivial.

Next we prove that Algorithm **PDcov** outputs r-approximate solutions. Consider the following linear program:

$$\begin{array}{lll} \text{(P)} & \min & wx \\ & \text{s.t.} & \alpha^k x \geq \beta^k \quad k \in \{1,\ldots,t\} \\ & & x \geq 0 \end{array}$$

where $\alpha^k x \geq \beta^k$ is the inequality used in the kth recursive call, and $t+1$ is the recursion depth. The dual is:

$$\begin{array}{lll} \text{(D)} & \max & \beta y \\ & \text{s.t.} & \sum_{k=1}^{t} \alpha_j^k y_k \leq w_j \quad j \in \{1,\ldots,n\} \\ & & y \geq 0 \end{array}$$

Examine the kth recursive call. Let z^k be the temporary partial solution at depth k. $\alpha^k x \geq \beta^k$ is a valid inequality with respect to $(\mathcal{F}, z^k)$, and, therefore, it is valid with respect to $\mathcal{F}$. Thus, $\textsc{Sol}(\mathcal{F}) \subseteq \textsc{Sol}(\text{P})$, and $\textsc{Opt}(\text{P}) \leq \textsc{Opt}(\mathcal{F}, w)$. As we have seen before x is a feasible solution for $\mathcal{F}$, and, therefore, for P. Also, y is a feasible solution for the dual of P.

Let x^k be the solution returned by the kth recursive call. Also, let w^k be the weight vector, and let j be the chosen element at the k'th call. We prove by induction that $w^k x^k \leq r \sum_{l \geq k} y_l \beta^l$. First, for $k = t+1$, we have $w^{t+1} x^{t+1} = 0 = \sum_{l \geq t+1} y_l \beta^l$. For $k \leq t$ we have,

$$\begin{aligned} w^k x^k &= (w^{k+1} + y_k \alpha^k) x^k \\ &= w^{k+1} x^{k+1} + y_k \alpha^k x^k \qquad (4) \\ &\leq r \sum_{l \geq k+1} y_l \beta^l + y_k r \beta^k \qquad (5) \\ &= r \sum_{l \geq k} y_l \beta^l \end{aligned}$$

where (4) is due to the fact that $w_j^{k+1} = 0$, and (5) is implied by the induction hypothesis, and the r-effectiveness of the inequality $\alpha^k x \geq \beta^k$. Finally,

$$wx = w^1 x^1 \leq r \sum_{l \geq 1} y_l \beta^l \leq r \cdot \textsc{Opt}(\text{P}) \leq r \cdot \textsc{Opt}(\mathcal{F}, w)$$

and we are done.

We remark that the value of y_k depends on the coefficients of the valid inequality $\alpha^k x \geq \beta$. That is, we can use the valid inequality $\rho \cdot \alpha^k x \geq \rho \cdot \beta$, for any $\rho > 0$, instead of using $\alpha^k x \geq \beta$, provided that the value of y_k is divided by ρ. In fact, by choosing the appropriate value of ρ, we can always ensure that $y_k = 1$. This fact is used in the sequel.

4.2 A Local Ratio Framework for Covering Problems

As was demonstrated in Section 3 the typical step of a local ratio algorithms involves the construction of a "good" weight function. Algorithm **LR-ST** used a weight function such that any minimal Steiner tree is 2-approximate with respect to it. In [4] Bar-Yehuda have defined this notion of goodness in the context of covering. The definition is given in our terminology.

Definition 3 ([4]) *Given a covering problem $(\mathcal{F}, w, z)$, a weight function δ is called* r-effective *with respect to $(\mathcal{F}, z)$, if $\forall j \in z, \delta_j = 0$, and every minimal feasible solution x with respect to $(\mathcal{F}, z)$ satisfies $\delta x \leq r \cdot \textsc{Opt}(\mathcal{F}, \delta, z)$.*

We prefer the following equivalent (yet more practical) definition.

Definition 4 *Given a covering problem $(\mathcal{F}, w, z)$, a weight function δ is called* r-effective *with respect to $(\mathcal{F}, z)$, if $\forall j \in z, \delta_j = 0$, and there exists β such that every minimal feasible solution x with respect to $(\mathcal{F}, z)$ satisfies: $\beta \leq \delta \cdot x \leq r\beta$. In this case we say that β is a* witness *to δ's r-effectiveness. If this is true for* any *integral feasible solution δ is called* fully r-effective.

We remark that we require $\delta_j = 0$ for every $j \in z$ since in general we deal with inequalities with respect to $(\mathcal{F}, z)$ and not with respect to $\mathcal{F}$. If $z = \emptyset$ we say that δ is r-effective with respect to $\mathcal{F}$.

Obviously, by assigning $\beta = \delta x^*$, where x^* is an optimal solution, we get that the first definition implies the latter. For the other direction, note that $\beta \leq \delta x^*$.

A local ratio algorithm for a covering problem works as follows. First, construct an r-effective weight function δ, such that $w - \delta \geq 0$ and there exists some j for which $w_j = \delta_j$. Such a weight function is called *w-tight*. Subtract δ from the weight function w. Add all zero weight elements to the partial solution z. Then, recursively solve the problem with respect to $(\mathcal{F}, w - \delta, z)$. When the empty set becomes feasible (or, when z becomes feasible with respect to $\mathcal{F}$) the recursion terminates. Finally, remove unnecessary elements from the temporary solution by performing a reverse deletion phase.

Algorithm **LRcov** is a generic local ratio approximation algorithm for covering problems. (The initial call is **LRcov**$(\emptyset, w)$.) The main difference between the algorithm from [4] and the one given here is that in the latter the augmentation of the temporary solution is done one element at a time. By doing this we have the option not to include zero weight elements which do not contribute to the feasibility of the partial solution z. When using the algorithm from [4] such elements are removed during the reverse deletion phase (called *removal loop* in [4]). In order to simulate the algorithm from [4] when using Algorithm **LRcov** we can add zero weight elements one by one. This is due to the fact that $\delta = 0$ is r-effective for all $r \geq 1$.

Algorithm LRcov(z, w)

1. If $\emptyset \in \textsc{Sol}(\mathcal{F}, z)$ return $\emptyset$
2. Construct a w-tight weight function δ which is r-effective w.r.t. $(\mathcal{F}, z)$
3. Let $j \notin z$ be an index for which $\delta_j = w_j$
4. $x \leftarrow \text{LRcov}(z \cup \{j\}, w - \delta)$
5. If $x \notin \textsc{Sol}(\mathcal{F}, z)$ then $x \leftarrow x \cup \{j\}$
6. Return x

Proving that Algorithm **LRcov** returns minimal feasible solutions with respect to $(\mathcal{F}, z)$ is essentially identical to proving that Algorithm **PDcov** returns minimal feasible solutions (see Section 4.1). Thus, we need only to prove that Algorithm **LRcov** outputs an r-approximate solution.

We prove by induction on the recursion that Algorithm **LRcov** returns an r-approximation with respect to $(\mathcal{F}, w, z)$. At the recursion basis, $\emptyset$ is an optimal solution. Otherwise, for the inductive step, examine x at the end of the recursive call. By the induction hypothesis $x \setminus \{j\}$ is an r-approximation with respect to $(\mathcal{F}, w - \delta, z \cup \{j\})$. Moreover, due to the fact that $w_j - \delta_j = 0$,

x is r-approximate with respect to $(\mathcal{F}, w - \delta, z)$. Finally, by the r-effectiveness of δ and the Local Ratio Theorem we get that x is an r-approximate solution with respect to $(\mathcal{F}, w, z)$ as well.

4.3 Equivalence

It is not hard to see that Algorithm **PDcov** and Algorithm **LRcov** share the same structure. Both algorithms, in each recursive call, modify the weights, add a zero-weight element to z, and solve the problem recursively. (Indeed, the proofs that both supply a feasible minimal solution are the same.) The difference between the two is that Algorithm **PDcov** uses r-effective inequalities, while Algorithm **LRcov** constructs r-effective weight functions. The following lemma shows that an r-effective valid inequality and an r-effective weight function are one and the same.

Lemma 2 *$\alpha x \geq \beta$ is an r-effective inequality if and only if α is an r-effective weight function with β as a witness.*

Proof. Let $\alpha x \geq \beta$ be an r-effective inequality. By definition every minimal feasible solution x satisfies: $\beta \leq \alpha x \leq r\beta$. Thus, α is an r-effective weight function. On the other hand, let α be an r-effective weight function with a witness β. Due to the r-effectiveness of α every minimal feasible solution x satisfies $\beta \leq \alpha x \leq r\beta$. Therefore, $\alpha x \geq \beta$ is an r-effective inequality. ■

We remark that when using an r-effective weight function δ, Algorithm **LRcov** does not need the know the value of the witness to δ's r-effectiveness. In fact, it can be NP-hard to calculate this value. The same goes for Algorithm **PDcov**. We do not have to know the value of the RHS of an r-effective inequality $\alpha x \geq \beta$. This is demonstrated in Section 4.4.4.

By Lemma 2 the use of a valid inequality can be simulated by utilizing the corresponding weight function, and vice versa. Thus, the primal-dual schema and the local ratio technique converge on standard applications.

Corollary 2 *Algorithms **PDcov** and **LRcov** are identical. Moreover, the equivalence is constructive, i.e., any implementation of one can be transformed into an implementation of the other.*

In the analysis of Algorithm **PDcov** we compared the integral primal solution x to a dual solution y in order to prove that the former is r-approximate. Recall that y was not a dual solution to the original program. We have defined a new program, called P, that contains the valid inequalities that were used by the algorithm, and the primal solution was compared to the dual of P. Clearly, the best approximation ratio we can hope for using this approach is the *integrality gap* of P. Thus, one can check whether an analysis for an algorithm is tight by comparing the performance ratio given by the analysis to the integrality gap of P. Now, consider the set of weight functions that were used by an implementation of algorithm **LRcov**. The corresponding inequalities would be the constraints of P. Thus, one can check whether an analysis of a local ratio algorithm is tight by calculating the integrality gap of P as well.

4.4 Applications of the Frameworks for Covering Problems

When trying to approximate an minimization problem we need to address several issues, which depend on the combinatorial structure of the problem at hand. First and foremost, we need to construct valid r-effective inequalities, or r-effective weight functions. Also, we need to use them such that the algorithm terminates in polynomial time. The algorithms for covering problems make use of the fact that you can add a zero weight element to the temporary partial solution, and, by that, reduce the size of the problem. This ensures that they work in polynomial time because there are no more than n iterations. Also, this allows us to use inequalities or weight functions which are r-effective with respect to the current problem, but are not necessarily r-effective with respect to the original problem. Many covering problems were approximated by making use of this mechanism (e.g., the feedback vertex set problem [2], and network design problems [25]). This is demonstrated in the sequel. Namely, we illustrate how Algorithms **PDcov** and **LRcov** can be used to construct and analyze approximation algorithms for covering problem. Note that when an algorithm is presented it is not given in full detail. We only describe the valid inequalities or weight functions needed in order to implement it using one of the generic algorithms.

Many approximation algorithms for covering problems use only one type of inequalities or weight functions. Such algorithms rely on the fact that when an instance is modified (or when an element is added to z, in our terminology) the resulting instance is still an instance of the same covering problem. For example, when Algorithm **LR-ST** contracts an edge the resulting instance is still an instance of the *Steiner tree* problem. Bertsimas and Teo [12] call an IP-formulation that satisfies this property *reducible*. Thus, in such cases, it is enough to describe and analyze an inequality or weight function with respect to the original set of constraints $\mathcal{F}$.

4.4.1 Steiner Tree and Other Network Design Problems

Let $\mathcal{F}$ be a set of constraints for the *Steiner tree* problem (e.g., the inequalities in program (ST)). Consider the instance $(\mathcal{F}, z)$ for some vector z. Recall that the elements (i.e., edges) in z are assumed to be taken into the solution. Thus, an instance $(\mathcal{F}, z)$ contains components on which there are connectivity demands. Baring this in mind it is not hard to see that Algorithm **LR-ST** (that is given in Section 3) can be viewed as an implementation of Algorithm **LRcov**. In each recursive call the algorithm uses the weight function $\delta(e) = \epsilon \cdot \tau(e)$, where $\epsilon = \min_e \{w(e)/\tau(e)\}$, and then contracts a zero weight edge. (Recall that $\tau(e)$ is the number of terminals incident to e.) This contraction can be represented by adding the edge e to z.

While Algorithm **LR-ST** can be viewed as an implementation of Algorithm **LRcov**, Algorithm **PD-ST** is not an implementation of Algorithm **PDcov**. For starters Algorithm **PD-ST** is iterative and not recursive. Also, it raises several dual variables in each iteration, and not one. However, as demonstrated in Section 4.1, when summing up the inequalities that correspond to the dual variables that are raised in an iteration we get Inequality 3, which is 2-effective. Therefore, it is enough to raise a single dual variable corresponding to Inequality 3 in each recursive call of Algorithm **PDcov**.

Algorithm **PD-ST** is a special case of an algorithm for *constrained forest* problems that was presented by Goemans and Williamson [24]. Given a graph $G = (V, E)$, a function $f : 2^V \to \{0, 1\}$, and a non negative weight function w on the edges Goemans and Williamson [24] have considered the following integer program:

$$\begin{array}{lll} \min & \sum_{e \in E} w_e x_e & \\ \text{s.t.} & \sum_{e \in \delta(S)} x_e \geq f(S) & \forall S,\ \emptyset \subsetneq S \subsetneq V \\ & x_e \in \{0, 1\} & \forall e \in E \end{array}$$

where $\delta(S)$ denotes the set of edges having exactly one endpoint in S. They have presented a $2 - \frac{2}{|A|}$-approximation algorithm, where $A = \{v : f(v) = 1\}$, for the case where f is *proper*.[5]

[5] A function f is *proper* if (1) $\forall S \subsetneq V,\ f(S) = f(V \setminus S)$; and (2) $\forall S, T,\ S \cap T = \emptyset,\ f(S \cup T) \leq \max\{f(S), f(T)\}$.

In [25] Goemans and Williamson have shown that the same algorithm outputs 2-approximate solution in the case of *downwards monotone* functions.[6] Williamson et al. [36] have generalized this algorithm for the class of *uncrossable* functions[7], and used this generalization to present a multi-phase primal-dual $2f_{\max}$-approximation algorithm for general proper functions, where $f_{\max} = \max_S f(S)$. They reduced the problem to a sequence of hitting set problems, and applied the primal-dual approximation algorithm for *uncrossable* functions to each subproblem. Thus, the solution to the original problem is the union of the solutions of the subproblems. Consequently, Goemans et al. [23], improved the approximation ratio to $2\mathcal{H}(f_{\max})$, where $\mathcal{H}$ is the harmonic function. (For more details see [25].)

Bertsimas and Teo [12] show that Inequality 3 is 2-effective even when f is *uncrossable*. Thus, all the above algorithms can be implemented using Algorithm **PDcov**. Moreover, because τ is a 2-effective weight function, all of them can be explained by local ratio means using Algorithm **LRcov**. In fact, the multi-phase primal-dual algorithms from [36, 23] can be analyzed as multi-phase local ratio algorithms. In [5] Bar-Yehuda et al. present the algorithm from [24] in local ratio terms, and, in particular, show that τ is 2-effective for *proper* and *downwards monotone* functions.

4.4.2 Generalized Hitting Set

The *generalized hitting set* problem is defined as follows. Given a collection of subsets S of a ground set E, a non negative weight $w(s)$ for every set $s \in S$, and a non negative weight $w(u)$ for every element $u \in E$, find a minimum-weight collection of objects $C \subseteq E \cup S$, such that for all $s \in S$, either there exists $u \in C$ such that $u \in s$, or $s \in C$. As in the *hitting set* problem our objective is to hit all the sets in S by using elements from E. However, in this case, we are allowed not to cover a set s, provided that we pay a tax $w(s)$. The hitting set problem is the special case where the tax is infinite for all sets. The generalized hitting set problem can be formalized as follows:

$$\begin{array}{lll} \min & \sum_{u \in E} w(u)x_u + \sum_{s \in S} w(s)x_s & \\ \text{s.t.} & \sum_{u \in s} x_u + x_s \geq 1 & \forall s \in S \\ & x_t \in \{0,1\} & \forall t \in E \cup S \end{array}$$

where $x_u = 1$ if and only if u is in the cover, and $x_s = 1$ if and only if s is not hit.

Observe that paying the tax $w(s)$ is required only when s is not hit. Thus, the inequality $\sum_{u \in s} x_u + x_s \geq 1$ is a Δ-effective inequality for any set $s \in S$, where $\Delta = \max\{|s| : s \in S\}$. The corresponding Δ-effective weight function is:

$$\delta(t) = \begin{cases} \epsilon, & t \in \{s\} \cup s \\ 0, & \text{otherwise} \end{cases}$$

Thus, a Δ-approximation algorithm can be constructed using one of the frameworks.

A linear time Δ-approximation algorithm can be obtained by extending the Δ-approximation algorithm for hitting set [6]. First, we use all the above inequalities (weight functions) in an arbitrary order; then a zero weight minimal feasible solution can be found: pick all zero weight elements and all the sets which are not hit by some zero weight element. This would be a Δ-approximate

[6] A function f is *downwards monotone* if $f(S) = 1$ implies $f(S') = 1$ for all nonempty $S' \subseteq S$.

[7] A function f is *uncrossable* if (1) $\forall S \subsetneq V,\ f(S) = f(V \setminus S)$; and (2) if S, T are intersecting sets such that $f(S) = f(T) = 1$ then either $f(S \setminus T) = f(T \setminus S) = 1$, or $f(S \cap T) = f(S \cup T) = 1$.

solution. When $\Delta = 2$ we get a special case called the *generalized vertex cover* problem, for which Hochbaum [29] present an $O(nm \log \frac{n^2}{m})$ 2-approximation algorithm.

We remark that the above inequalities remain Δ-effective if we use any value between 1 and Δ as x_s's coefficient. Analogously, any value between ϵ and $\Delta \cdot \epsilon$ is acceptable for $\delta(s)$.

4.4.3 Feedback Vertex Set in Tournaments

A *tournament* is an orientation of a complete (undirected) graph, i.e., it is a directed graph with the property that for every unordered pair of distinct vertices $\{u, v\}$ it either contains the arc (u, v) or the arc (v, u), but not both. The *feedback vertex set in tournaments* problem is the following. Given a tournament and a weight function w on its vertices, find a minimum-weight set of vertices whose removal leaves a graph containing no directed cycles.

It is not hard to verify that a tournament contains a directed cycle if and only if it contains a *triangle*, where a triangle is a directed cycle of length 3. Thus, we may restrict our attention to triangles, and formulate the problem as follows:

$$\begin{array}{lll} \text{(FVST)} & \min & \sum_{v \in V} w_v x_v \\ & \text{s.t.} & \sum_{v \in T} x_v \geq 1 \quad \forall \text{ triangle } T \\ & & x_v \in \{0, 1\} \qquad \forall v \in V \end{array}$$

We say that a triangle is *positive* if all of its vertices have strictly positive weights. Clearly, the set of all zero-weight vertices is an optimal solution (of zero weight) if and only if the tournament contains no positive triangles. Thus we obtain a local ratio 3-approximation algorithm by means of the following 3-effective weight function. Let $\{v_1, v_2, v_3\}$ be a positive triangle, and let $\epsilon = \min\{w(v_1), w(v_2), w(v_3)\}$. Define:

$$\delta(v) = \begin{cases} \epsilon & v \in \{v_1, v_2, v_3\}, \\ 0 & \text{otherwise.} \end{cases}$$

The maximum cost, with respect to δ, of a feasible solution is clearly at most 3ϵ, while the minimum cost is at least ϵ, since every feasible solution must contain at least on of v_1, v_2, v_3. The corresponding 3-effective inequality is $x_{v_1} + x_{v_2} + x_{v_3} \geq 1$.

Note that *any* feasible solution is 3-approximate with respect to δ (not only minimal solutions). Equivalently, the inequality $x_{v_1} + x_{v_2} + x_{v_3} \leq 3$ holds for *any* feasible solution. Thus, the weight function and inequality are fully r-effective.

4.4.4 Feedback Vertex Set

A set of vertices in an undirected graph is called a *feedback vertex set* (FVS for short) if its removal leaves an acyclic graph (i.e., a forest). In other words, the set must cover all cycles in the graph. The *feedback vertex set* problem is: given a vertex-weighted graph to find a minimum weight FVS.

In [2] Bafna et al. have presented a local ratio 2-approximation algorithm for the *feedback vertex set* problem. Their algorithm can be implemented using Algorithm **LRcov** with the following weight functions. If G contains a *semi-disjoint* cycle[8] C let $\epsilon = \min_{v \in C} w(v)$, and use the weight function

$$\delta_1(v) = \begin{cases} \epsilon & v \in C, \\ 0 & \text{otherwise,} \end{cases}$$

otherwise use the weight function $\delta_2(v) = \epsilon \cdot (\deg(v) - 1)$, where ϵ is defined to be $\min_{v \in V} \{w(v)/(\deg(v) - 1)\}$. It is not hard to see that if there exist a *semi-disjoint* cycle C then δ_1 is 1-effective. Bafna et al. [2] showed that δ_2 is 2-effective in graphs that (1) do not contain *semi-disjoint* cycles; and (2) $\deg(v) \geq 2$ for every $v \in V$. In order implement this algorithm within the primal-dual schema one should use the following valid inequalities: $\sum_{v \in C} x_v \geq 1$ in case G contains a *semi-disjoint* cycle C, and $\sum_{v \in V} (\deg(v) - 1) \cdot x_v \geq |E| - |V| + 1$, otherwise.

Another 2-approximation algorithm is due to Becker and Geiger [11]. In [4] Bar-Yehuda indicated that their algorithm can be restated in local ratio terms with the weight function $\delta(v) = \deg(v)$, which is 2-effective. It can be shown that the corresponding 2-effective inequality is $\sum_{v \in V} \deg(v) x_v \geq |E| - |V| + 1 + \tau$, where τ is the cardinality of the smallest FVS in G. Therefore, a primal-dual analysis to this algorithm can be given by using Algorithm **PDcov**. It is important to note that we do not need to know the value τ in order to execute the algorithm. In fact, this value is NP-hard to compute.

Chudak et al. [15] have explained both algorithms using primal-dual arguments, and added a third 2-approximation algorithm which is similar to the one from [2]. We present it as an implementation of Algorithm **PDcov**. That is, we show which inequality to use in each recursive call. Choose an *end-block*[9] B, and use the following inequality: $\sum_{v \in V} (\deg(v) - 1) x_v \geq |E| - |V| + 1$. The corresponding weight function is

$$\delta(v) = \begin{cases} \epsilon \cdot (\deg(v) - 1) & v \in B, \\ 0 & \text{otherwise.} \end{cases}$$

A local ratio implementation of all three algorithms, and a detailed analysis of the one from [11] can be found in [5].

5 Minimization Frameworks

The recursive algorithms for covering problems can be divided into three primitives: the recursion base, the way that an instance is modified before a recursive call, and the way in which the solution returned by a recursive call is fixed. The frameworks for covering problems heavily rely on the fact that the set of constraints $\mathcal{F}$ is monotone. In each recursive call a zero-weight element j is added to z, and by that a new instance is created. A solution to this new instance is returned by the recursive call, and then fixed in a very straightforward manner—add j to the solution if it is not feasible. By adding an new element to z in each recursive call the algorithm is bound to arrive to the recursion base which is the empty instance, for which the empty set is always a minimal optimal solution. In this section we present a more general framework that can explain many algorithms that do not fall within the scope of our generic algorithms for covering. This is done by means of extending each of the three primitives mentioned above. This time we start with the local ratio framework.

[8] A cycle C is *semi-disjoint* if $\exists x \in C$ such that $\deg(u) = 2$ for every vertex $u \in C \setminus \{x\}$.

[9] An *end-block* is a biconnected component containing at most one articulation point.

5.1 Local Ratio

Algorithm **LRcov** modifies the current instance by assuming that a zero-weight element is taken into the solution (i.e., by adding a zero-weight element to z). This can be done because in covering problems adding a zero-weight element to the solution is never a bad move. However, in non covering problem, a solution containing this element may not even exist. Also, in non boolean problems, there are several possible assignments for a zero-weight variable. Thus, we need extend the algorithms for covering by considering more ways in which to modify the instance.

After each recursive call Algorithm **LRcov** fixes, if necessary, the solution returned in order to turn it into a "good" solution, i.e., into a minimal solution. This is done because the algorithm uses weight functions that are r-effective. It turns out that an algorithm may use weight functions for which good solutions are solutions that satisfy a certain property different from *minimality*. In fact, this property can be, simply, *the solutions returned by the algorithm*. We refer to such weight functions as r-effective with respect to a property $\mathcal{P}$. Clearly, in such cases, the algorithm may be forced to fix the solution returned by a recursive call in a way that is very different from simply adding a single element in case the current solution is not feasible.

In [7] Bar-Yehuda and Even developed a $(2 - \frac{\log_2 \log_2 n}{2 \log_2 n})$-approximation algorithm for *vertex cover* which is partly based on local-ratio. Their algorithm starts with a local ratio phase that removes short odd cycles from the graph, and then continues to the next phase that finds approximate solutions for graphs that do not have short odd cycles. This can be explained by a variant of Algorithm **LRcov** in which the recursion base (Line 1) is replaced by the invocation of an approximation algorithm that only works for a certain kind of inputs and returns r-approximate minimal solutions. (The solution need not be minimal if the weight functions used are fully r-effective.)

Our framework can be described as follows. In each recursive call the algorithm constructs and uses a weight function, and modifies the instance. Then, it recursively solves the problem on the new instance and the new objective function. Afterwards, it fixes the solution returned. The recursion base is performed if an instance satisfies some property $\mathcal{Q}$.

The algorithm uses the following three subroutines:

- **Base**$(\mathcal{F}, w)$: Given a problem instance that satisfies $\mathcal{Q}$ returns an r-approximate solution.

  In most cases, the base of the recursion is simply to return an empty set when the instance is an empty. However, the recursion base may be a complicated operation, as shown in Section 5.4.1, or in [7].

- **Modify**$(\mathcal{F}, w)$: Modifies the current instance by assigning values to zero-weight variables, and then removing them.

  In most algorithms the modification involve a single zero-weight element. A more complicated modification is described in Section 5.4.2.

- **Fix**(x'): Given an r-approximate solution x' for the instance **Modify**$(\mathcal{F}, w)$, returns an r-approximate solution x for the instance $(\mathcal{F}, w)$ satisfying some property $\mathcal{P}$. (Note that each recursive call may use a different property.)

  Typically, when dealing with covering problems a solution is fixed by adding a zero-weight element if necessary. In cases where the property $\mathcal{P}$ is not *minimality* the transformation may be somewhat more involved as demonstarted in Sections 5.4.3.

The algorithm is as follows.

Algorithm LRmin$(\mathcal{F}, w)$

1. If $\mathcal{F}$ satisfies $\mathcal{Q}$ return **Base**$(\mathcal{F}, w)$
2. Construct a weight function δ which is r-effective with respect to a property $\mathcal{P}$, such that $w - \delta \geq 0$
3. $\mathcal{F}' \leftarrow$ **Modify**$(\mathcal{F}, w - \delta)$
4. $x' \leftarrow$ **LRmin**$(\mathcal{F}', w - \delta)$
5. $x \leftarrow$ **Fix**(x')
6. Return x

The analysis of Algorithm **LRmin** is similar to the analysis given for Algorithm **LRcov**. We prove that the algorithm returns an r-approximate solution by induction on the recursion. The recursion base is trivial because Subroutine **Base** returns r-approximate solutions by definition. For the inductive step, consider the solution x' that was returned by the recursive call. By the inductive hypothesis x' is r-approximate with respect to $(\mathcal{F}', w - \delta)$. Due to Subroutines **Modify** and **Fix** x is r-approximate with respect to $(\mathcal{F}, w - \delta)$ and satisfies property $\mathcal{P}$. Furthermore, δ is r-effective with respect to $\mathcal{P}$. Thus, by the Local Ratio Theorem x is also r-approximate with respect to $(\mathcal{F}, w)$.

5.2 Primal-Dual

Algorithm **PDmin** is our primal-dual approximation algorithm. It uses the same three primitives that are used by Algorithm **LRmin**.

Algorithm PDmin$(\mathcal{F}, w)$

1. If $\mathcal{F}$ satisfies $\mathcal{Q}$ return **Base**$(\mathcal{F}, w)$
2. Construct an inequality $\alpha x \geq \beta$ which is r-effective with respect to a property $\mathcal{P}$, such that $w - \alpha \geq 0$
3. $\mathcal{F}' \leftarrow$ **Modify**$(\mathcal{F}, w - \alpha)$
4. $x' \leftarrow$ **PDmin**$(\mathcal{F}', w - \alpha)$
5. $x \leftarrow$ **Fix**(x')
6. Return x

Before presenting our primal-dual analysis, we need to set the following two conditions.

1. Subroutine **Modify** modifies an instance such that any valid inequality with respect to the modified instance is also valid with respect to the original instance.

2. Subroutine **Base** returns a solution whose weight is no more than r times the optimal solution of the LP-relaxation of an IP-formulation of its input.

These conditions make sure that a primal-dual analysis is possible. The exact role of the conditions will become apparent shortly. We remark that both conditions are met in the case of our primal-dual framework for covering problems.

First, due to Subroutine **Fix**, Algorithm **PDmin** returns feasible primal solutions. We show that it returns r-approximate solutions. We do that by generalizing the analysis of Algorithm **PD-cov**. Let $(\mathcal{F}_k, w_k)$ be the instance given to the kth recursive call, let x^k be the solution returned by the kth recursive call, and let $t+1$ be the recursion depth. Consider the following linear program:

$$\begin{array}{lll} \text{(P)} & \min & wx \\ & \text{s.t.} & \alpha^k x \geq \beta^k \quad k \in \{1,\ldots,t+m\} \\ & & x \geq 0 \end{array}$$

where, $\alpha^k x \geq \beta^k$ for $k \leq t$ is the inequality used in the kth recursive call, and $\left\{\alpha^k x \geq \beta^k : k \in \{t+1,\ldots,t+m\}\right\}$ is a set of linear constraints that describes the base instance $\mathcal{F}_{t+1}$. By Condition 1, P is a relaxation of $\mathcal{F}$, and therefore $\textsc{Opt}(\text{P}) \leq \textsc{Opt}(\mathcal{F}, w)$.

Next, we build a solution to the dual of P, called D. Let P_k be the LP that we get from P by discarding the first $k-1$ inequalities, and changing the objective function to $w_k x$. Also, let D_k be the dual of P_k. Consider the base instance $(\mathcal{F}_{t+1}, w_{t+1})$. By Condition 2 Subroutine **Base** returns a solution x^{t+1} whose weight is no more than r times the optimal solution of P_{t+1} (the LP-relaxation of an IP-formulation of $(\mathcal{F}_{t+1}, w_{t+1})$). Thus, $w_{t+1}x^{t+1}$ is bounded by r times the value of y^*, where y^* is an optimal solution to D_{t+1}. We define a vector y as follows:

$$y_k = \begin{cases} 1 & k \leq t, \\ y^*_{k-t} & k > t \end{cases}$$

That is, y consists of t 1s, and then the coefficients of y^*. Let y^k be the vector that consists of $t-k$ 1s followed by the coefficients of y^*. That is, y^k is a vector that contains the last $m+t-k$ coefficients of y. We prove by induction that y^k is a solution to D_k. And because $y = y^1$, and $\text{D} = \text{D}_1$ we get that y is a dual solution to D. At the base of the recursion, $y^{t+1} = y^*$ is an optimal solution to D_{t+1} by definition. For the inductive step, we assume that y^{k+1} is a solution to D_{k+1}, and prove that y^k is a solution to D_k. First, we claim that $(0, y^{k+1})$ (a vector that consists of a zero followed by the coefficients of y^{k+1}) is a feasible solution to D_k. To see this notice that a packing of constraints from P_{k+1} is also a packing of constraints from P_k. Moreover, $y^k = (1, y^{k+1})$ is also a packing of constraints from P_k, since $w_k = w_{k+1} + \alpha$.

We prove by induction that $w^k x^k \leq r \sum_{l \geq k} y_l \beta^l$. For $k = t+1$, by Condition 2 we know that $w^{t+1}x^{t+1} \leq r \cdot \sum_{l \geq t+1} y_l \beta^l$. For $k \leq t$ we have,

$$\begin{aligned} w^k x^k &= (w^{k+1} + \alpha^k)x^k \\ &= w^{k+1}x^{k+1} + y_k \alpha^k x^k && (6) \\ &\leq r \sum_{l \geq k+1} y_l \beta^l + y_k r \beta^k && (7) \\ &= r \sum_{l \geq k} y_l \beta^l \end{aligned}$$

where (6) is stems from the fact that Subroutine **Modify** removes only zero-weight elements from an instance, and (7) is due to the induction hypothesis, and the r-effectiveness of the inequality $\alpha^k x \geq \beta^k$. Finally,

$$wx = w^1 x^1 \leq r \sum_{l \geq 1} y_l \beta^l \leq r \cdot \textsc{Opt}(\text{P}) \leq r \cdot \textsc{Opt}(\mathcal{F}, w)$$

and we are done.

5.3 Discussion

The only unknown elements in the framework for covering are the r-effective inequalities (weight functions). That is, in order to construct an algorithm for a covering problem one has to find the appropriate inequalities (weight functions) and the rest is determined by the framework. The task of designing an algorithm may be much more complicated when one chooses to use the framework given in this section. For starters one has to come up with a suitable and polynomial implementation of Subroutines **Base**, **Modify**, and **Fix**. Also, the resulting algorithm must reach the recursion base in polynomial time. Intuitively, after finding an r-effective inequality (weight function) we must ask ourselves the following question: How should we remove zero-weight elements? We must be able to remove zero-weight elements in a way that enables us to later fix the solution returned by the recursive call. A good answer to this question is an implementation of Subroutines **Modify**, and **Fix**. Note that, as in the covering setting, our generic algorithms may use a different type of inequality (weight function) in each recursive call. Moreover, they may use a different property in each recursive call. However, this may require to implement several versions of Subroutines **Modify**, and **Fix**. When using a non trivial recursive base, we can look at the primal-dual (local ratio) phase of the algorithm as a clean-up phase whose output is an instance of a certain type that we know how to solve by Subroutine **Base**.

The minimization frameworks can be applied on a large family of algorithms. They can be used in cases of non covering problems as demonstarted in Section 5.4.2 on a problem called *minimum 2-satisfiability*. They can be used to analyze algorithms that have a non standard recursion base. A 2.5-approximation algorithm for *feedback vertex set in tournaments* that has a non standard base is given in Section 5.4.1. The frameworks can be used to explain algorithms that do not use r-effectiveness with respect to *minimality*, and use a non standard instance modification. They can also be used on problems whose solutions are non boolean. An algorithm using a non standard instance modification that approximates a non boolean *bandwidth trading* problem is given in Section 5.4.3. Another example of an algorithm approximating a non boolean problem is an algorithm by Guha et al. [26] for *capacitated vertex cover*. A local ratio version of this primal-dual algorithm can be found in [5].

Another important point is that due to Lemma 2 Algorithm **LRmin** and **PDmin** are equivalent. Thus, putting aside for a minute the conditions made in the primal-dual case, the equivalence between the two paradigms that was shown with respect to algorithms for covering problems continue to hold even in a more general setting. Having said that we must return to the issue of the conditions made in the primal-dual case. As we have seen the analysis of Algorithm **LRmin** did not require the conditions. This is because the local ratio technique uses a *local* approach. A typical local ratio advancement step is local in the sense that it can be analyzed independently from the rest of the algorithm (see also [4]). Therefore, local ratio algorithms tend to be recursive, and their analyses inductive. The analysis of algorithm **LRmin** is no exception to this rule. On the other hand, primal-dual analyses use a more *global* approach. A typical primal-dual algorithm is iterative, and rely (unnecessarily) on a predetermined LP-formulation. In order to bound the weight of the primal solution a dual solution must be constructed, and the ratio between the weight of the primal and the weight of the dual determines the performance ratio of the algorithm. This global approach makes it difficult for primal-dual to analyze Algorithm **PDmin**, and therefore we had to add the two conditions. These conditions make sure that we are able to build a program that relaxes the original problem, and to construct the desired dual solution.

The discussion above suggests that the local approach is stronger than the global approach, or at least easier to use, because we do not have to show that the conditions are satisfied when using

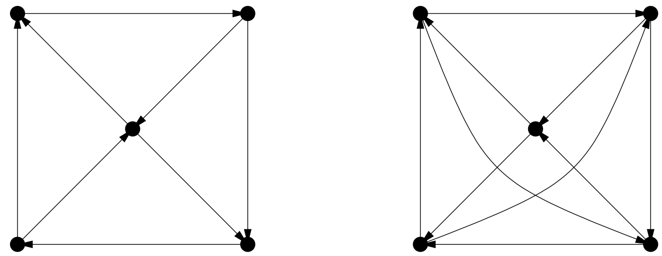

Figure 1: Forbidden sub-tournaments.

local ratio. However, the global approach of primal-dual has its advantages too. For example, the algorithms for covering use reverse deletion in order to transform the solution into a minimal solution. However, in several primal-dual algorithms for covering problems (e.g., [24]) it can be shown that this transformation can be done in ways other than reverse deletion. This can also be done when using local ratio, but it is less natural.

5.4 Applications of the Minimization Frameworks

5.4.1 Feedback Vertex Set in Tournaments

Cai et al. [14] present a 2.5-approximation algorithm for *feedback vertex set in tournaments* problem (that was defined in Section 4.4.3). The algorithm is divided into two parts: a local ratio phase that disposes of certain *forbidden* sub-tournaments, and an algorithm that finds an optimal solution in any tournament that does not contain forbidden sub-tournaments. The forbidden sub-tournaments are shown in Figure 1 (where the two arcs not shown in T_1 may take any direction). The local ratio phase employs the following fully 2.5-effective weight function. Let F be a set of five positive-weight vertices inducing a forbidden sub-tournament and define:

$$\delta(v) = \begin{cases} \epsilon & v \in F, \\ 0 & \text{otherwise.} \end{cases}$$

where $\epsilon = \min\{w(v) : v \in F\}$. δ is fully 2.5-effective since the cost of every feasible solution is clearly at most $5 \cdot \epsilon$, whereas the minimum weight is at least $2 \cdot \epsilon$ (as every set of four vertices in F contain a triangle). After removing at least one vertex from every forbidden sub-tournament using local ratio, the problem can be solved optimally on the remaining graph. (A detailed presentation of the local ratio part of this algorithm can be found in [5].)

This algorithm can be seen as a specific implementation of Algorithm **LRmin** in which the Subroutines **Modify** and **Fix** are standard, and Subroutine **Base** is the algorithm that solve the problem on tournaments that do not contain forbidden sub-tournaments. (In other words, it can be implemented by a version of Algorithm **LRcov** in which the recursion base is modified as mentioned above.)

Using our primal-dual framework, this algorithm can be also analyzed using primal-dual arguments. This can be done by using 2.5-effective inequalities of the form $\sum_{u \in F} x_u \geq 2$, where F is a set of five positive-weight vertices inducing a forbidden sub-tournament. Clearly, these inequalities are valid with respect to the original instance, and therefore Condition 1 is satisfied. In [14], Cai et al. show that the integrality gap of program (FVST) (see Section 4.4.3) is 1 in the case of tournaments that does not contain forbidden sub-tournaments.[10] Thus, the implementation of Subroutine **Base** satisfies Condition 2.

5.4.2 Minimum Weight 2-satisfiability

Given a 2CNF formula φ on the variables $x_1, \ldots, x_n$, and a weight function w on the variables, the weight of a truth assignment $x \in \{0,1\}^n$ is $\sum_{i=1}^n w_i x_i$. The *minimum weight* 2*-satisfiability* problem (or min-2SAT for short) is to find a minimum weight truth assignment $x \in \{0,1\}^n$ which satisfies φ, or determine that no such assignment exists. We formulate min-2SAT as follows:

$$\begin{array}{lll} \min & \sum_{i=1}^n w_i x_i & \\ \text{s.t.} & x_i + x_j \geq 1 & \forall (x_i \vee x_j) \in \varphi \\ & x_i - x_j \geq 0 & \forall (x_i \vee \overline{x}_j) \in \varphi \\ & -x_i - x_j \geq -1 & \forall (\overline{x}_i \vee \overline{x}_j) \in \varphi \\ & x_i \in \{0,1\} & \forall i \in \{1, \ldots, n\} \end{array}$$

Gusfield and Pitt [27] presented an $O(mn)$ time 2-approximation algorithm for min-2SAT, where n is the number of variables and m is the number of clauses. Though they did not use local ratio arguments explicitly, their algorithm can be easily analyzed via the local ratio technique. At about the same time Hochbaum et al. [30] have presented a 2-approximation algorithm for the *two variables per constraint integer programming* problem (2VIP) that generalizes min-2SAT. Later, Bar-Yehuda and Rawitz [9] have presented a local ratio 2-approximation algorithm for 2VIP that is more efficient than the algorithm from [30]. On the special case of min-2SAT this algorithm becomes a variant of the Gusfield and Pitt algorithm [27]. We remark that min-2SAT can be approximated using an approximation preserving reduction to vertex cover [28, pp. 131–132].

First, we can check whether φ is satisfiable by using the algorithm from [19]. Therefore, we may assume that φ is satisfiable. In order to construct a 2-approximation algorithm we need to find 2-effective inequalities (or weight functions). Given a literal ℓ, let $T(\ell)$ be the set of variables which must be assigned TRUE whenever ℓ is assigned TRUE. (Constructing $T(\ell)$ for some literal ℓ can be done efficiently by using constraint propagation.) Let x_i, x_j and x_k be variables such that $x_j \in T(x_i)$ and $x_k \in T(\overline{x}_i)$. For such variables the inequality $x_j + x_k \geq 1$ is valid. Note that one can get inequalities of this form by summing up the appropriate inequalities from the above LP formulation of 2SAT. Moreover, it is not hard to see that this inequality is fully 2-effective. Instead of using these inequalities one at a time, we can use an inequality of the form

$$\sum_{x_j \in T(x_i)} a_j x_j + \sum_{x_k \in T(\overline{x}_i)} b_k x_k \geq \beta$$

where all the a_js and b_ks are non negative and $\beta = \sum_j a_j = \sum_k b_k$. This is due to the fact that this inequality is a linear combination of inequalities of the form $x_j + x_k \geq 1$.

Let $\alpha x \geq \beta$ be such an inequality in which $\beta = \min\{\sum_{x_j \in T(x_i)} w_j, \sum_{x_k \in T(\overline{x}_i)} w_k\}$. Assume without loss of generality that $\sum_{x_i \in T(x_1)} w_i \leq \sum_{x_j \in T(\overline{x}_1)} w_j$. Observe that if we subtract α from the objective function, assigning TRUE to all literals in $T(x_i)$ is free of charge. It can be shown that this partial assignment does not change the satisfiability of the formula. That is, if φ' is the formula we get by performing this zero-weight partial assignment to the variables of a formula φ,

[10] Cai et al. [14] actually prove a stronger claim. They show that in tournaments that does not contain forbidden sub-tournaments both primal and dual programs have integral optimal solutions whose weights are the same.

φ' is satisfiable if and only if φ is satisfiable. After performing this instance modification the rest of the assignment is found recursively.

Thus, the primal-dual implementation of the algorithm is as follows. At the recursion base we return an empty assignment on the empty formula. Clearly, Condition 2 is satisfied in this case. If the formula φ is not empty, we pick a variable x_i, and construct an inequality $\alpha x \geq \beta$ as shown above. Note that such inequalities are valid with respect to the original instance, and therefore Condition 1 is satisfied. We call Subroutine **Modify** that in this case constructs a zero-weight partial assignment for φ, and creates a new formula φ'. Then, we recursively solve the problem on φ'. Afterwards, Subroutine **Fix** combines the assignment for φ' that was returned and the partial assignment that was constructed by Subroutine **Modify**. For the local ratio implementation, it is enough to notice that that α is a fully 2-effective weight function. (For more details see [9].)

5.4.3 A Bandwidth Trading Problem

Bhatia et al. [13] have studied the following *bandwidth trading* problem. In this problem we are given a set of machine *types* $\mathcal{T} = \{T_1, \ldots, T_m\}$ and a set of *jobs* $J = \{1, \ldots, n\}$. Each machine type T_i is defined by two parameters: a time interval $I(T_i)$ during which it is *available*, and a weight $w(T_i)$, which represents the weight of allocating a machine of this type. Each job j is defined by a single time interval $I(j)$ during which it must be processed. We say that job j *contains* time t if $t \in I(j)$. A given job j may be *scheduled feasibly* on a machine of type T if type T is available throughout the job's interval, i.e., if $I(j) \subseteq I(T)$. A *schedule* is a set of machines together with an assignment of each job to one of them. It is *feasible* if every job is assigned feasibly and no two jobs with intersecting intervals are assigned to the same machine. The weight of a feasible schedule is the total cost of the machines it uses, where the weight of a machine is defined as the weight associated with its type. The goal is to find a minimum-weight feasible schedule. We assume that a feasible schedule exists. (This can be checked easily.)

Bhatia et al. [13] have presented a primal-dual 3-approximation algorithm for this *bandwidth trading* problem. A detailed local ratio analysis of their algorithm can be found in [5]. This algorithm constructs weight functions or inequalities that are r-effective weight functions with respect to a property $\mathcal{P}$ different from *minimality*, and modifies solution returned by a recursive call in a rather elabotare manner.

We present the algorithm in local ratio terms.

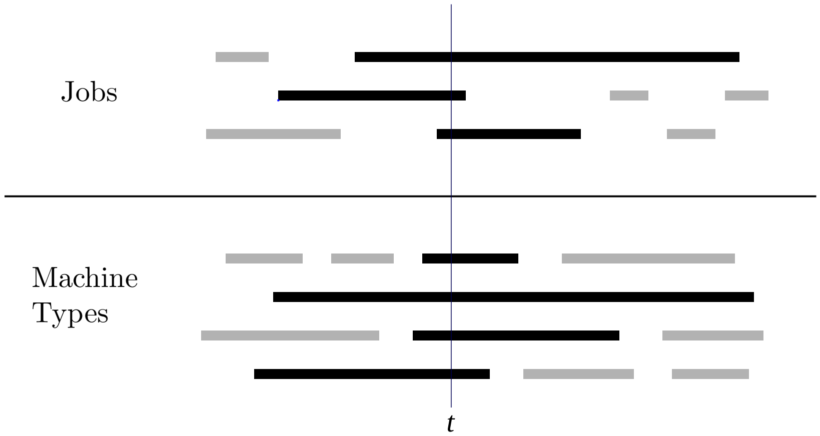

Figure 2: Jobs containing time t (top, dark), and machine types available at time t (bottom, dark).

Algorithm BT$(\mathcal{T}, J, w)$

1. If $J = \emptyset$, return $\emptyset$
2. Let t be a point in time contained in a maximum number of jobs, and let $\mathcal{T}_t$ be the set of machine types available at time t (see Figure 2)
3. Let $\epsilon = \min\{w(T) \,:\, T \in \mathcal{T}_t\}$
4. Define the weight function $\delta(T) = \begin{cases} \epsilon & T \in \mathcal{T}_t, \\ 0 & \text{otherwise}, \end{cases}$

   /∗ Subroutine **Modify** ∗/
5. Let $\mathcal{T}'_t = \{T \,:\, T \in T_t,\ w(T) = \delta(T)\}$
6. Let $J' = \{j \in J \,:\, \exists T \in \mathcal{T}'_t,\ I(j) \subseteq I(T)\}$
7. $S' \leftarrow \mathbf{BT}(\mathcal{T} \setminus \mathcal{T}'_t, J \setminus J', w - \delta)$

   /∗ Subroutine **Fix** ∗/
8. Extend S' to all J by allocating $|J'|$ machines and scheduling one job from J' on each. Job $j \in J'$ is assigned to a machine of type $T \in \mathcal{T}'_t$ such that $I(j) \subseteq I(T)$.
9. Transform S' into a new schedule S in a manner that is discussed below
10. Return S

To complete the description of the algorithm we need to describe the transformation of S' to S referred to in Line 9. Instead, we just point out two facts relating to this transformation. (The details of the transformation appear in [13] and also in [5].)

1. For all machine types T, S does not use more machines of type T than S'.
2. Let k be the number of jobs containing time t (Line 2). The number of machines used by S whose types are in $\mathcal{T}_t$ is at most $3k$.

Based on these fact, we show that Algorithm **BT** is a specific implementation of Algorithm **LRmin** that returns 3-approximate solutions. By Fact 1, $w'(S) \leq w'(S')$, where $w' = w - \delta$, and therefore

S to is 3-approximate with respect to w'. Thus, Subroutines **Modify** and **Fix** work as required. (Subroutine **Base** is standard in this case.) Also, by Fact 2, $\delta(S) \le 3k\epsilon$, and because there are k jobs containing time t—each of which can be scheduled only on machines whose types are in $\mathcal{T}_t$, and no two of which may be scheduled on the same machine—the optimum cost is at least $k\epsilon$. Thus, S is 3-approximate with respect to δ.

We now turn to the primal-dual analysis. Bhatia et al. [13] have formulated the bandwidth trading problem by the following program:

$$
\begin{array}{lll}
\min & \sum_{i=1}^{n} w(T_i)x_i & \\
\text{s.t.} & \sum_i y_{ij} \ge 1 & \forall j \in J \\
 & x_i - \sum_{j \in J(t)} y_{ij} \ge 0 & \forall T_i \in \mathcal{T},\ \forall t \in E \cap I(T_i) \\
 & x_i \in \mathbb{N} & \forall T_i \in \mathcal{T} \\
 & y_{ij} \in \{0,1\} & \forall T_i \in \mathcal{T}, j \in J
\end{array}
$$

where,

- x_i represents the number of machines allocated of type T_i.
- $y_{ij} = 1$ if and only if job j is assigned to machine type T_i. Note that y_{ij} is defined only if $I(j) \subseteq I(T_i)$, where i is of type T.
- E is the set of endpoints of job intervals.
- $J(t) = \{j \,:\, t \in I(j)\}$.

In order to turn Algorithm **BT** into a primal-dual algorithm, we use the inequality $\delta \cdot x \ge k\epsilon$ (instead of using the weight function δ). Similarly to the local ratio case, it can be shown that this version of Algorithm **BT** is a specific implementation of Algorithm **PDmin**. To see that Condition 1 is satisfied notice that the above inequality is valid with respect to the original instance. This is because if there are k jobs whose interval contains time t, then at least k machines whose types belong to T_t must be allocated. Condition 2 holds trivially.

We remark that our primal-dual analysis is slightly different from the analysis in [13]. Specifically, their algorithm uses similar but not identical inequalities that can be described as linear combinations of inequalities from the above formulation.

6 Maximization Problems

In [3] Bar-Noy et al. used the local-ratio technique to develop constant factor approximation algorithms for various resource allocation and scheduling problems. They also presented primal-dual algorithms for these problems. This was the first time a local ratio or primal-dual approximation algorithm for a natural maximization problem was presented. In this section we present two equivalent generic approximation algorithms for maximization problems that can be used to analyze the algorithms from [3]. We demonstarte this on one of the problems that was discussed in [3] called *interval scheduling*. Also, we show that our generic algorithms can explain the exact optimization (or, 1-approximation) algorithm for the *longest path in a DAG* problem.

6.1 The Frameworks

Before describing the generic algorithms we address the issue of r-effectiveness in the context of maximization. We discuss the issue in terms of weight functions, but a similar discussion can be made in terms of inequalities. Recall that δ is r-effective with respect to a property $\mathcal{P}$ if there exists β such that $\beta \leq \delta x \leq r \cdot \beta$ for every solution x that satisfies $\mathcal{P}$. In the maximization setting it is more convenient to consider the following equivalent definition. δ is r-effective with respect to a property $\mathcal{P}$ if there exists β such that $\frac{\beta}{r} \leq \delta x \leq \beta$ for every solution x that satisfies $\mathcal{P}$. This way it is clear that any feasible solution that satisfies $\mathcal{P}$ is r-approximate with respect to δ.

Our frameworks are recursive and and work as follows. If the instance is empty return an the empty set (the recursion terminates). Otherwise, construct a weight function (inequality) that is r-effective with respect to some property $\mathcal{P}$. Subtract the weight function (coefficients of inequality) from the objective function w. Remove all non positive weight elements from the instance. Then, recursively solve the problem with respect to the new instance and weights. Upon returning from the recursive call the solution returned is fixed such that it satisfies $\mathcal{P}$. It is important to note that the weight function (inequality) is built in a way that ensures the ability to fix the solution returned by the recursive call.

We start with our generic local ratio approximation algorithm for maximization problems—Algorithm **LRmax**. (The initial call is **LRmax**$(\{1, \ldots, n\}, w)$.) A recursive call of Algorithm **LR-max** considers the instance that is induced by the set of elements N that corresponds to the set of positive weight elements. It starts with the construction of a weight function δ. Then, a recursive call is made on the instance that is induced by the objective function $w - \delta$ and the set $N \setminus N^-$, where N^- is a set that contains non positive weight elements with respect to $w - \delta$. Subroutine **Fix** is used to fix the solution returned by adding only zero weight elements with respect to $w - \delta$. The resulting solution satisfies property $\mathcal{P}$.

Algorithm LRmax(N, w)

If $N = \emptyset$, return $\emptyset$
Construct a weight function δ which is r-effective w.r.t. $(\mathcal{F}, N)$ and $\mathcal{P}$
Let $N^- \subseteq \{j : w_j - \delta_j \leq 0\}$
$x' \leftarrow$ **LRmax**$(N \setminus N^-, w - \delta)$
$x \leftarrow$ **Fix**$(x, w - \delta)$
Return x

We prove by induction that Algorithm **LRmax** returns an r-approximate solutions with respect to (N, w). In the base case, $\emptyset$ is an optimal solution. For the inductive step, examine x at the end of the recursive call. By the induction hypothesis x' is r-approximate with respect to $(N \setminus N^-, w - \delta)$. Moreover, due to the fact that $w_j - \delta_j \leq 0$ for any $j \in N^-$, x is r-approximate with respect to $(N, w - \delta)$. (Recall that Subroutine **Fix** adds only zero weight elements with respect to $w - \delta$.) Finally, x satisfies $\mathcal{P}$ due to Subroutine **Fix**, therefore by the r-effectiveness of δ with respect to $\mathcal{P}$, and the Local Ratio Theorem we get that x is r-approximate respect to (N, w) as well.

Algorithm **PDmax** is very similar to Algorithm **LRmax**. Obviously, Algorithm **PDmax** uses inequalities instead of weight functions. Also, as in the minimization case, we assume that the inequalities that are used by the algorithm are valid with respect to the original set of constraints $\mathcal{F}$. This condition is imperative to the construction of a feasible dual solution.

Algorithm PDmax(N, w)

1. If $N = \emptyset$, return $\emptyset$
2. Construct a valid inequality $\alpha^k x \leq \beta^k$ which is r-effective w.r.t. $(\mathcal{F}, N)$ and $\mathcal{P}$
3. Let $N^- \subseteq \{j : w_j \leq \alpha_j\}$
4. $x' \leftarrow$ **PDmax**$(N \setminus N^-, w - \alpha)$
5. $x \leftarrow$ **Fix**$(x, w - \alpha)$
6. Return x

We show that Algorithm **PDmax** returns r-approximate solutions. Let a notation with subscript k denote the appropriate object in the kth iteration, and let $t+1$ be the recursion depth. Consider the following linear program:

$$\begin{array}{lll} \text{(P)} & \min & wx \\ & \text{s.t.} & \alpha^k x \leq \beta^k \quad k \in \{1, \ldots, t\} \\ & & x \geq 0 \end{array}$$

where, $\alpha^k x \leq \beta^k$ is the inequality used in the kth recursive call. Every feasible solution satisfies the constraints in P, namely $\textsc{Sol}(\mathcal{F}) \subseteq \textsc{Sol}(\text{P})$. Thus, $x \in \textsc{Sol}(\text{P})$, and $\textsc{Opt}(\text{P}) \geq \textsc{Opt}(\mathcal{F}, w)$.

Consider the dual of P:

$$\begin{array}{lll} \text{(D)} & \min & \sum_{k=1}^{t} \beta^k y_k \\ & \text{s.t.} & \sum_{k=1}^{t} \alpha_j^k y_k \geq w_j \quad j \in \{1, \ldots, n\} \\ & & y \geq 0 \end{array}$$

We claim that $y = 1$ is a feasible solution to D. To do that we conceptually add the following between Line 2 and Line 3: $y_k \leftarrow 1$. Clearly, the resulting dual solution is $y = 1$. In terms of the dual solution, elements leave the set N only when their corresponding dual constraint is satisfied. Algorithm **PDmax** terminates when the current instance is empty, namely when $N = \emptyset$. Therefore, at termination all dual constraints are satisfied.

We prove by induction that $w^k x^k \geq \frac{1}{r} \sum_{l \geq k} y_l \beta^l$. At the induction basis, $0 = w^{t+1} x^{t+1} \geq \frac{1}{r} \sum_{l \geq k} y_l \beta^l = 0$. For $k \leq t$ we have

$$w^k x^k = (w^{k+1} + \alpha^k) x^k = w^{k+1} x^{k+1} + y_k \alpha^k x^k \geq \frac{1}{r} \cdot \sum_{l \geq k+1} y_l \beta^l + \frac{\beta^k}{r} = \frac{1}{r} \cdot \sum_{l \geq k} y_l \beta^l$$

where the second equality is due to the fact that Subroutine **Fix** uses only zero-weight elements, and the inequality is implied by the induction hypothesis, and the r-effectiveness of the kth inequality. Therefore, $wx = w^1 x^1 \geq \frac{1}{r} \sum_{l \geq 1} y_l \beta^l \geq \frac{1}{r} \cdot \textsc{Opt}(\text{P}) \geq \frac{1}{r} \cdot \textsc{Opt}(\mathcal{F}, w)$.

It is important to notice that the maximization case is different from the minimization case. In the latter we keep the weights non-negative, while in the former weights are allowed to be negative. Moreover, the weight function in the maximization case is expected to be non-positive when the algorithm terminates. This means, in primal-dual terms, that the dual solution is initially not feasible, and its feasibility is improved during the execution of the algorithm. Also, at termination, the negative coordinates of the weight function correspond to the non-tight dual constraints. This difference makes life a bit more complicated when dealing with maximization problem. Specifically, in the minimization case, the weight function δ is constructed such that it satisfies two conditions:

(1) $\delta \leq w$, and (2) there exists an element j for which $w_j = \delta_j$. In the maximization case, the second condition is satisfied, but the first is not.

We remark that, in order to simply the presentation, our maximization frameworks are not as general as our minimization frameworks. Namely, the maximization frameworks use a limited version of Subroutine **Modify** that simply removes non-positive elements from the instance. (This is characteristic to algorithm for *packing* problems.) Also, they do not use a version Subroutine **Base** at all.

6.2 Applications of the Frameworks for Maximization

6.2.1 Interval Scheduling

As mentioned before, in [3] Bar-Noy et al. present local ratio approximation algorithms for several resource allocation and scheduling problems that can be explained by our frameworks. We demonstrate this by analyzing one of the algorithms from [3] that approximates a problem called *interval scheduling*. Bar-Noy et al. also present primal-dual algorithms for the same problems. However, in order to do so they modified the original algorithms. We show that there is no need to change the algorithms in order to supply a primal-dual analysis.

In the interval scheduling problem we are given a set of *activities*, each requiring the utilization of a given *resource*. The activities are specified as a collection of sets $\mathcal{A}_1, \ldots, \mathcal{A}_m$. Each set represents a single activity: it consists of all possible *instances* of that activity. An instance $I \in \mathcal{A}_i$ is defined by the following parameters:

1. A half-open time interval $[s(I), e(I))$ during which the activity will be executed. $s(I)$ and $e(I)$ are called the *start-time* and *end-time* of the instance; And,
2. The weight $w(I) \geq 0$ gained by scheduling this instance of the activity.

A *schedule* is a collection of instances. It is feasible if it contains at most one instance of every activity, and at most one instance for all time instants t. In the interval scheduling problem our goal is to find a schedule that maximizes the total weight accrued by instances in the schedule.

The interval scheduling problem can be formulated by means of an integer program on the boolean variables $\{x_I : I \in \mathcal{A}_i, 1 \leq i \leq m\}$.

$$\begin{array}{lll} \max & \sum_I w(I) x_I & \\ \text{s.t.} & \sum_{I: s(I) \leq t < e(I)} x_I \leq 1 & \forall t \\ & \sum_{I: I \in \mathcal{A}_i} x_I \leq 1 & \forall i \in \{1, \ldots, m\} \\ & x_I \in \{0,1\} & \forall i \; \forall I \in A_i, \end{array}$$

The 2-approximation algorithm for interval scheduling from [3] can be viewed as an application of Algorithm **LRmax**. In order to describe it as such, we need to show how to construct a weight function δ that is 2-effective with respect to some property $\mathcal{P}$, which elements are removed from the instance (i.e., which elements are taken into N^-), and how to fix the solution returned by the recursive call (i.e., describe Subroutine **Fix**). Let J be an instance with minimum end-time, and

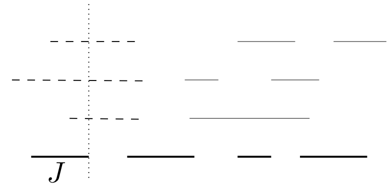

Figure 3: $J, \mathcal{A}(J), \mathcal{I}(J)$: heavy lines represent $A(J)$, dashed lines represent $\mathcal{I}(J)$.

let $\mathcal{A}(J)$ and $\mathcal{I}(J)$ be the activity to which instance J belongs and the set of instances intersecting J (including J), respectively. (See Figure 3.) Define

$$\delta(I) = \begin{cases} w(J) & I \in \mathcal{A}(J) \cup \mathcal{I}(J), \\ 0 & \text{otherwise.} \end{cases}$$

We show that δ is 2-effective with respect to some property $\mathcal{P}$. We say that a feasible schedule S is J-maximal if either it contains J or J cannot be added to S without rendering it infeasible. It is not hard to verify that the weight of every J-maximal schedule with respect to δ is at least $w(J)$ and no more than $2 \cdot w(J)$. (Notice that a feasible schedule contains no more than two instances from $\mathcal{A}(J) \cup \mathcal{I}(J)$.) Now, the elements that are taken into N^- are all non positive elements with respect to $w - \delta$. Finally, we describe Subroutine **Fix**. Let S' be the schedule returned by the recursive call. If $S' \cup \{J\}$ is a feasible solution return $S = S' \cup \{J\}$. Otherwise, return $S = S'$. Clearly, S is J-maximal.

As mentioned before, Bar-Noy et al. [3] also presented primal-dual algorithms that are slightly different from their local ratio algorithms. In terms of the interval scheduling problem they modified the original algorithm by using a different 2-effective weight function:

$$\delta'(I) = \begin{cases} w(J) & I = J, \\ \frac{1}{2} w(J) & I \in \mathcal{A}(J) \cup \mathcal{I}(J) \setminus \{J\}, \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding inequality is $\frac{1}{2}\sum_{I \in \mathcal{I}(J)} x_I + \frac{1}{2}\sum_{I \in A(J)} x_I \leq 2$. Note that this inequality is a linear combination of two inequalities from the above integer program. The original algorithm can be explained by the 2-effective inequality $\sum_{I \in \mathcal{I}(J) \cup \mathcal{A}(J)} x_I \leq 2$. The difference between δ and δ' (or between their corresponding inequalities) is the ratio between the weight of J and the weights of the other instances in $\mathcal{A}(J) \cup \mathcal{I}(J)$. In fact, any value between 1 and 2 is acceptable.

6.2.2 Longest Path in a DAG

The *longest path* problem is, given an arc-weighted directed graph $G = (V, A)$ and two distinguished vertices s and t, find a simple path from s to t of maximum *length*, where the *length* of a path is defined as the sum of weights of its arcs. For general graphs (either directed or undirected) the problem is NP-hard [22], but for *directed acyclic graphs* (DAGs) it is solvable in linear time by a Dijkstra-like algorithm that processes the nodes in topological order. The problem of finding the longest path in a DAG (also called *critical path*) arises in the context of PERT (Program Evaluation and Review Technique) charts. For more details see [16, page 538] or [18, pp. 138–142].

In this section we show that the above mentioned linear time algorithm can be seen as an implementation of Algorithms **LRmax** and **PDmax**. (A detailed, but slightly different, local ratio analysis of this algorithm is given in [5].) We allow negative arc weights, and we assume, without loss of generality, that every vertex is reachable from s. (Otherwise, simply delete all vertices that are unreachable from s.) We also assume that the vertices of G were topologically sorted, and that t is the last vertex in this topological sort. Instead of solving the original problem we solve the following more general problem. Namely, instead of searching for a longest path from s to t we would like to find the longest path from some vertex in a set S to t without using arcs within S. Note that in the original problem $S = \{s\}$. Also, if $s \in S$ and for all $u \in S$ the longest path from s to u is of length zero, then the problem is equivalent to the original problem as well.

Consider a cut $(S, \bar{S})$ such that $s \in S, t \in \bar{S}$, and there is no arc that leaves $\bar{S}$ and enters S. Note that if we take the first k vertices in the topological sort we get such a cut. We define the following weight function:

$$\delta(e) = \begin{cases} \epsilon & e \in S \times \bar{S} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, any path from s to t must cross the cut $(S, \bar{S})$ exactly once, thus δ is fully 1-effective. Equivalently, the equality $\sum_{e \in S \times \bar{S}} x_e = 1$ is valid. Having defined a suitable weight function or equality we continue with a description of the algorithm. We describe a recursive call of the algorithm using local ratio terms. Let v be the vertex which is the first in $\bar{S}$ according to the topological sort. Let $\epsilon = \max_{u \in S} \{w(u, v)\}$, and let $e = (u, v)$ be an arc such that $u \in S$ and $w(u, v) = \epsilon$. If $v = t$ then return a path containing u and t. Otherwise, solve the problem recursively on $(G, S \cup \{v\}, w)$. Now, let $v_1, \ldots, v_\ell$ be the path returned. If $v_1 = v$ then return the path $u, v_1, \ldots, v_\ell$, otherwise return $v_1, \ldots, v_\ell$.

Acknowledgments

We thank Guy Even, Eran Mann, Seffi Naor, and especially Ari Freund for helpful comments and discussions.

References

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[2] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

[3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Shieber. A unified approach to approximating resource allocation and schedualing. *Journal of the ACM*, 48(5):1069–1090, 2001.

[4] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

[5] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. *Local Ratio*: A unified framework for approximation algorithms. Submitted, 2003.

[6] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[7] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[8] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 732–741, 2002.

[9] R. Bar-Yehuda and D. Rawitz. Efficient algorithms for bounded integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.

[10] R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local-ratio technique. In *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, volume 2129 of *LNCS*, pages 24–35, 2001.

[11] A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.

[12] D. Bertsimas and C. Teo. From valid inequalities to heuristics: A unified view of primal-dual approximation algorithms in covering problems. *Operations Research*, 46(4):503–514, 1998.

[13] R. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. In *30th International Colloquium on Automata, Languages, and Programming*, volume 2719 of *LNCS*, pages 751–766, 2003.

[14] M. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM Journal on Computing*, 30(6):1993–2007, 2001.

[15] F. A. Chudak, M. X. Goemans, D. S. Hochbaum, and D. P. Williamson. A primal-dual interpretation of recent 2-approximation algorithms for the feedback vertex set problem in undirected graphs. *Operations Research Letters*, 22:111–118, 1998.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[17] G. B. Dantzig, L. R. Ford, and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, 1956.

[18] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[19] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

[20] A. Freund and D. Rawitz. Combinatorial interpretations of dual fitting and primal fitting. In *Workshop on Approximation and Online Algorithms*, volume 2909 of *LNCS*, pages 137–150, 2003.

[21] T. Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 86:213–231, 1998.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[23] M. X. Goemans, A. Goldberg, S.Plotkin, D. Shmoys, E. Tardos, and D. P. Williamson. Improved approximation algorithms for network design problems. In *5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 1994.

[24] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[25] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Hochbaum [28], chapter 4.

[26] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering with applications. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 858–865, 2002.

[27] D. Gusfield and L. Pitt. A bounded approximation for the minimum cost 2-SAT problem. *Algorithmica*, 8:103–117, 1992.

[28] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problem*. PWS Publishing Company, 1997.

[29] D. S. Hochbaum. A framework of half integrality and good approximations. Manuscript, UC berkeley, 1997.

[30] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993.

[31] H. Karloff. *Linear Programming*. Progress in Theoretical Computer Science. Birkhäuser Boston, 1991.

[32] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization; Algorithms and Complexity*. Prentice-Hall, Inc., 5th edition, 1982.

[33] R. Ravi and P. Klein. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *3rd MPS Conference on Integer Programming and Combinatorial Optimization*, pages 39–56, 1993.

[34] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2nd edition, 2001.

[35] D. P. Williamson. The primal dual method for approximation algorithms. *Mathematical Programming*, 91(3):447–478, 2002.

[36] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problems. *Combinatorica*, 15:435–454, 1995.