How well can Primal-Dual and Local-Ratio algorithms perform? [*]

Allan Borodin David Cashman[†]
Avner Magen

Department of Computer Science, University of Toronto

March 9, 2011

Abstract

We define an algorithmic paradigm, the stack model, that captures many primal dual and local ratio algorithms for approximating covering and packing problems. The stack model is defined syntactically and without any complexity limitations and hence our approximation bounds are independent of the P vs NP question. Using the stack model, we bound the performance of a broad class of primal dual and local ratio algorithms and supply a $(\log n+1)/2$ inapproximability result for set cover, a 4/3 inapproximability for min Steiner tree, and a 0.913 inapproximability for interval scheduling on two machines.

1 Introduction

The primal dual and local ratio schemas for approximation algorithms are two fundamental algorithm design techniques. The use of the primal dual schema is pervasive, having been applied to give good approximation algorithms for several basic NP-hard combinatorial optimization problems including set cover, vertex cover, numerous network design problems, facility location and k-median, Steiner forest, and many others. The origins of the primal dual schema can be found in the Bar Yehuda and Even [14] algorithm for the weighted Vertex Cover problem. The re-introduction of the enhanced primal dual schema leading to its current importance is due to the works of Agarwal, Klein and Ravi [2] and Goemans and Williamson [43].

The list of problems tackled successfully by primal dual and local ratio algorithms is long and impressive. For many of these problems the methods achieve the best known approximation (see [61, 13] and references therein). But just how far can these methods go? The goal of this paper is to shed some light on this question. We do so by introducing a syntactic computational model devoid of complexity considerations that will simulate a broad class of primal dual algorithms, and that will be amenable to analysis that bounds its performance. This model is what we call the *stack model* and it will be described in detail in Section 2. The stack model more directly

[*]A preliminary version of this paper appears in the Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), pp 943-955, July 2005.

[†]Current address: Altera Corporation, 151 Bloor St. W., Suite 200, Toronto, Canada

captures (an iterative version of) a class of local-ratio algorithms as described in the survey of Bar-Yehuda et al [13], which is consistent with the work by Bar-Yehuda and Rawitz [17] who show that there is a strong connection between the two paradigms. However, as discussed in section 2.3, stack algorithms will not capture all such algorithms (even in the context of covering and packing problems).

We shall focus our attention on what is considered to be a common instantiation of the primal dual schema for covering problems. Let us be more specific: we begin with what Williamson [61] calls [1] the "general dual ascent primal dual algorithm". The same general schema is also articulated by Betsimas and Teo [22]. A substantial leap forward in terms of the applicability of the primal dual approach was made at the beginning of the nineties, with the introduction of the "reverse delete step" [43, 49, 56]. Roughly speaking, this is a phase that (by considering input items in the reverse order as they were added to the solution) guarantees that the output of the algorithm is a *minimal* feasible solution (with respect to set inclusion). This second phase is shown to be crucial in the analysis' of covering problems such as the minimum Steiner forest problem and the feedback vertex set problem [19], and others. An explanation of the role of the reverse delete step is given in [61].

In contrast to the extensive use of primal dual and local ratio algorithms for covering problems, the explicit use of either schema for packing problems seems to have only recently been articulated in Bar-Noy et al [11]. They consider the approximation of a very basic and general scheduling problem, *the job bandwidth scheduling problem* with many important special cases including interval scheduling, throughput maximization, the job interval scheduling problem $JISP$, and the knapsack problem. Independently, Berman and Das Gupta [21] provide an iterative version of essentially the same local search algorithm for many of the problems in Bar Noy et al [11]. For packing problems, the reverse delete stage is used to guarantee the feasibility of the solution. As shown in [11] and [21], the (two pass) local ratio technique can *optimally* solve the one machine weighted interval selection problem whereas the negative results in [23, 46, 4] provide general one pass models relative to which interval selection cannot be solved optimally. Following these papers, there have been a number of other applications ([3, 20, 12, 1, 5, 16]) of the local ratio method for packing problems.

Our desire to model and study the limitations of a particular algorithmic paradigm is, of course, part of a substantial history of such studies. We mention a few notable efforts in this regard. Perhaps the best known development is the Edmonds [35] (respectively, Korte and Lovász [51]) characterization of matroids (respectively, greedoids) in terms of *the natural* "best in" greedy algorithm optimally maximizing a linear function over independent (respectively, feasible) sets in a set system. An extension of this characterization for non linear functions over greedoids can be found in Dechter and Dechter [34]. Chvátal [32] provides complexity lower bounds for a model of branch and bound methods solving the knapsack problem. Representative of inapproximation results for restricted classes of algorithms, we mention the Khanna et al [48] study of locality gaps for local search methods applied to MAXSAT. Also of considerable note are the Arora, et al [9] integrality gaps for classes of LP relaxations and lift-and-project hierarchies for the vertex cover problem. Other integrality gap results pertaining to such methods can be found in [40, 41, 57, 42, 29].

Our work follows most closely and is an extension of the priority algorithm model of Borodin, Nielsen and Rackoff [23] and Davis and Impagliazzo [33]. In their pBT (for priority branching tree)

[1]Williamson defines the primal- dual method in terms of a general hitting set problem which in turn models a number of covering problems. (See the Appendix for the natural LP relaxation of the hitting set problem and the related primal dual schema.) The description of primal-dual in terms of the hitting set problem first appeared in chapter 4 (by Goemans and Williamson) of Hochbaum's [45] edited text.

extension of the priority model, Alekhnovich et al [4] provide complexity lower bounds for models of "simple" dynamic programming and backtracking as applied to the m-machine interval scheduling, the knapsack problem and SAT search problems. Buresh-Oppenheim, Davis and Impagliazzo [24] extend the pBT model to the pBP (for priority branching program) model so as to capture a wider class of dynmaic programming algorithms. In this paper, we introduce another extension of the priority model, the (priority) stack model, that captures many primal dual and local ratio algorithms for approximating covering and packing problems. The stack model is defined syntactically and without any complexity limitations and hence our approximation bounds are independent of the P vs NP question. Using the stack model, we bound the performance of a broad class of primal dual and local ratio algorithms and supply a $(\log n + 1)/2$ inapproximability result for set cover, a $4/3$ inapproximability for min Steiner tree, and a 0.913 inapproximability for interval scheduling on two machines. We are not aware of any previous formalization of the primal dual and local ratio methods.

2 The Stack Model and its relation to Primal Dual and Local Ratio algorithms

In this section we give a formal definition of our stack model and describe how it relates to certain basic forms of primal dual and local ratio algorithms. We emphasize that our stack model does not make any complexity assumptions and hence our inapproximation results are based on the syntactic structure of the algorithm and the restricted way by which such algorithms can access information about the input. Our focus at first is on covering problems, and we then treat the case of packing problems. A central object of this section is an integer program associated with a covering problem. We will make the standard assumption that we associate variables with the natural 0/1 variables for the problem. Specifically, we wish to minimize $\sum_i c_i x_i$ subject to the constraints $A\mathbf{x} \geq \mathbf{b}$, and the entries of the matrix A as well as the vectors $\mathbf{b}$ and $\mathbf{c}$ are all nonnegative. We shall use the set cover problem as a running example of a covering problem. In this example we have a universe U (known a-priori to the algorithm) and sets $S_1, \ldots, S_n \subset U$ where each set S_i has cost c_i. The integer program has 0/1 variables x_i that correspond with the sets S_i, and the goal is to minimize $\sum c_i x_i$ subject to the constraints $\sum_{i:j \in S_i} x_i \geq 1$ for each element $j \in U$.

2.1 The Stack Model for Covering Problems

The stack model is a two-pass algorithm, in which the first pass, the push phase, resembles an adaptive priority algorithm, as defined in [23] and formalized in [33]. Priority algorithms are one pass algorithms where irrevocable decisions (i.e. accept/reject for covering problems) are made for each input item. In contrast, stack algorithms push (rather than accept) and reject input items in the first pass and then the second pass, the pop phase, is simply used to ensure minimality of the solution (with respect to set inclusion). We now give a precise and self contained description. An input instance is a finite set of "input items". Without knowing the actual input instance, the algorithm first orders all possible input items by some ordering, say Π_1. This means that the algorithm must commit to a total ordering of all (infinitely many) possible items, and the actual input instance (i.e. a finite set of input items) then inherits the induced ordering. Alternatively, one can think of a function ordering every pair of possible items in a consistent (i.e., transitive) way. In our set cover example, knowing the given universe U, the algorithm must order all possible

```
Let $\mathcal{I}' \subset \mathcal{I}$ be the actual input set
While $\mathcal{I}'$ is not empty
    Determine a total ordering $\Pi$ on $\mathcal{I}$, the set of all possible remaining input items
    $next$ := index of input item $I$ in $\mathcal{I}'$ that comes first in the ordering
    As a function of all input items and decisions in previous iterations, make a decision
      as to whether to reject $I_{next}$ or push $I_{next}$ onto the stack
    Remove $I_{next}$ from $\mathcal{I}'$ and remove $I_{next}$ and all input items that proceed $I_{next}$ in $\Pi$ from $\mathcal{I}$
End While
$Sol$ := set of items on stack
While the stack is not empty
    $next$ := index of input item $I$ on top of stack
    Pop $I_{next}$ from stack
    remove $I_{next}$ from $Sol$ if $Sol - \{I_{next}\}$ is feasible
End While
```

Figure 1: The form of an adaptive stack algorithm for a covering problem.

input items of the form $(S, c(S))$ where $S \subset U$ and $c(S)$ is the positive real valued weight (or cost) of subset S. One can imagine ordering by cardinality of sets, or more generally by increasing (or decreasing) values of any real valued function f_1 on the weight and elements of each set. Next, the algorithm looks at the first item in the input instance and for a priority algorithm (respectively, a stack algorithm) either *accepts* or *rejects* the item (respectively, either *pushes* it onto a stack or *rejects* it). Using the information obtained from this first input item i_1, the algorithm then chooses a new total ordering Π_2 on the set of all possible remaining items (i.e., those that come after item i_1 in the initial ordering) which induces a new ordering on the actual set of remaining input items. For the set cover example, the algorithm can define an appropriate f_2 after looking at the first set considered (and has been accepted, rejected, or pushed onto the stack) and knowing which elements are now covered. Again, the first (unseen) input item (in the input instance) in this ordering is either accepted/rejected (for priority algorithms), or pushed onto the stack or *rejected* (for stack algorithms). This process continues for at most n iterations where n is the number of primal variables. At this point, for a priority algorithm (respectively, a stack algorithm) , the accepted items (respectively, the items pushed onto the stack) must contain a feasible solution. For stack algorithms, the algorithm the enters the "pop phase" in which items are popped from the stack (last in first out). A popped item will be *rejected* if the items that were popped and accepted, together with the items still on the stack, constitute a feasible solution; otherwise the popped item is *accepted*. It is easy to see that if at the beginning of the pop phase we have a feasible solution, we will have one at its termination. It is also clear that the resulting solution is minimal with respect to set inclusion.

We note that an observation in Davis and Impagliazzo [33] immediately shows that the pop stage can provably improve approximation ratios. Namely, while Dijkstra's algorithm produces a shortest path tree from a given source node s, they observe that no priority algorithm can accept exactly a single $s-t$ path. It follows that priority algorithms cannot produce an optimal $s-t$ path whereas Dijkstra's algorithm is easily implemented as a stack algorithm by using the pop phase to eliminate unnecessary edges in the shortest path tree. Note also that the shortest $s-t$ path problem is precisely the Steiner tree problem when there are two required nodes s and t.

If there is a fixed ordering of all possible input items (i.e. $\Pi_1 = \Pi_2 = \Pi_3 = \ldots$) that is used throughout the single phase of a priority algorithm (respectively, throughout the push phase of a stack algorithm) and is not changed during the process, we call this a *fixed order* priority (respectively, stack) algorithm. Without this restriction we say the ordering is *adaptive.* Pseudo code for an adaptive stack covering algorithm is presented in figure 1.

2.2 Primal-dual algorithms of an LP relaxation

We begin by formulating an abstraction of dual-ascent primal dual algorithms. The primal dual schema considers a Linear Programming relaxation of the integer program and its dual maximization problem and proceeds as follows:

> (i) Start with the feasible dual solution $\mathbf{y} = 0$, and an integral but infeasible solution $\mathbf{x} = 0$. All dual variables y_j are initially *unfrozen*; set iteration counter $\ell := 1$.
>
> (ii) Continuously increase each unfrozen dual variable y_j at some rate r_j^ℓ until one (or more) of the dual constraints becomes tight. (It is useful to think of a continuous procedure where a real variable t denotes "time" and the value of y_j at time t is $y_j(t) = \sum_{\ell=1}^{k-1}(t_\ell - t_{\ell-1})r_j^\ell + (t - t_{k-1})r_j^k$, for $t \in [t_{k-1}, t_k]$.)
>
> (iii) When a dual constraint becomes tight, the corresponding primal variable is set to 1. All the dual variables in this constraint then become *frozen.* [2]
>
> (iv) As long as the primal solution is not yet feasible and some dual variables are still unfrozen, increment ℓ and go back to (ii). If all variables are frozen and the primal solution is not yet feasible, then report failure.
>
> (v) Go over the primal variables in the solution (i.e. those that were assigned value 1) in reverse order, removing variables "greedily". That is, the value of a variable changes from 1 to 0 if this doesn't render the solution infeasible.

Step (ii) above is the heart of the algorithm, in the sense that its unspecified details (i.e. the different rates) must be determined in order to transform the schema into a specific well-defined algorithm. Notice that all other steps are totally specified. To make the schema useful the rates $\{r_j^\ell\}$ must be efficiently computed. We note that in Williamson's [61] schema, the rate is uniform for a set of unfrozen dual variables U_ℓ corresponding to a subset T_ℓ of violated primal constraints and zero otherwise. That is, $r_j^\ell = 1$ for the chosen dual variables $y_j \in U_\ell$. In this setting, in order to completely specify the algorithm, it is only left to define the set of violated constraints to be used at each iteration. An important observation is that in some sense our algorithms may be as strong as the recipe of choice of rates. If there is no limitation on this recipe optimality is always achievable. More precisely, there is always a "clever" choice of violated constraints that will lead to an optimal solution. However such clever choices are in general inefficient to compute, and in fact may be as hard to find as the optimum in the first place. That is, once an optimal solution S is known, the dual constraints corresponding to primal variables in S can first be made tight and then all other primal variables will become redundant in the pop phase. This observation suggests

[2]We have chosen to allow stack algorithms to reject items during the push phase as this is consistent with the priority model and potentially adds power (in terms of approximation ratios) to the model. We can similarly add power to the primal dual schema by allowing such algorithms to reject items during the first stage.

that we must make some limitations on the recipe of the choice of rates. As is usually the case with definitions of syntactic model, we replace the "obvious" limitation of polytime computability of the rates with some information theoretic criteria that is formalized in Definition 2.

Returning to our set cover example, the dual variables in the natural LP relaxation are associated with elements of the ground set. The standard primal dual algorithm uniformly increases all the dual variables that do not occur in a tight equation (uncovered elements), until a new dual constraint (corresponding to a set) becomes tight, in which case this set joins the solution. Step (v), the reverse-delete phase, deletes a set if it is not needed, while following the reverse order.

2.3 Simulating Primal-dual (and local-ratio) algorithms with the Stack Model

Having defined the abstraction of the primal dual method as well as the stack model, we need to make the connection between them precise. A primal dual algorithm is based on a particular LP relaxation, and any strength or weakness it may have must be related to that particular relaxation; in particular, as observed in [61], the ratio guarantee of primal dual algorithms is never better than the integrality gap for *the specific linear program relaxation used by the algorithm*. In contrast, the stack model has no LP flavor to it at all! The weakness of the stack model is a result of the limited information that is revealed at different points of the execution of the algorithm. We therefore have to reconcile these two different frameworks.

Definition 1. Let $\mathcal{L}$ be a particular LP relaxation for an integer program with 0/1 variables. We say that an algorithm for the problem in the stack model is $\mathcal{L}$ compatible if
(i) the items of the stack model correspond to the variables of $\mathcal{L}$.
(ii) the information that is revealed in an item (say corresponding to the LP variable x_i) contains its cost (its coefficient c_i in the objective function) and associated coefficients (i.e. the $\{a_{ij}\}$ for variable x_i) in $\mathcal{L}$.

We also require the following restriction of primal dual algorithms (as a syntactic analogue to the efficiency of the algorithm in say choosing violated constraints).

Definition 2. A primal dual algorithm is *history based* if in all iterations ℓ, the dual increase rates r_j^ℓ are a function of the current primal variables in the solution and all the current values of the dual variables. In particular, the initial rates r_j^1 do not depend on the actual input.

We now get to the critical and perhaps subtle observation that is implicit in the stack model and that will be key in deriving lower bounds. If at any point an adversary *eliminates* a primal variable corresponding to a dual constraint that has not yet become tight, this has no effect on the preceding runtime history of the algorithm. An adversarial approach can therefore be developed from this limitation of the algorithm and may lead to inapproximability results. (The reader who is familiar with the adversarial analysis of online algorithms may find the above limitation resembling that of online algorithms.)

In Williamson's motivating example of the vertex cover problem and the more general hitting set problem (see Appendix) where each target set T_i has cardinality at most f (equivalently the f-frequency set cover problem), the basic f-approximation primal dual algorithm chooses an arbitrary violated constraint which in our statement of the schema allows us to set one rate $r_i = 1$ (and all others set to zero) corresponding to a set T_i which has not yet been hit. This is clearly a history-based primal dual algorithm and is consistent with a stack model in which the items are elements $e \in E$ where each element item e is represented by its weight c_e and by the sets in which it is contained. In our set-cover example, a stack model is compatible with respect to the standard LP relaxation if its input items are sets, and further, the representation of each set provides the cost of the set and the (names of the) elements that it contains.

In [31], Chudak et al provide primal dual (with reverse delete) interpretations for two related algorithms ([19], [10]) for the vertex feedback set problem, both providing 2-approximations. The Becker and Geiger [19] algorithm maintains a current set S of vertices still in the "remaining graph" (i.e. after isolated and degree 1 vertices have been deleted and other vertices have been placed into the vertex feedback set and then deleted) and then raises the corresponding dual variable y_S until some dual constraint $\sum_{S:v\in S} degree_S(v) y_S \leq w_v$ becomes tight. The corresponding primal variable v is then added to the vertex feedback set and removed. This is therefore a history-based primal dual algorithm compatible with a stack model in which the items correspond to vertices where the information in each vertex item v is the weight w_v of the item and the names of its adjacent vertices (i.e. the standard adjacency list representation). We note that this primal dual algorithm rejects items during the first stage (i.e. when it knows that a vertex is isolated or has degree 1) as such vertices cannot cut any remaining cycles. On the other hand, the algorithm of Bafna et al [10] needs to determine the vertices of a violated semi-disjoint cycle (i.e. a cycle that contains at most one vertex of degree greater than 2) so as to recompute edge weights and is not a history-based primal dual algorithm.

Claim 3. *Every primal dual algorithm based on LP relaxation $\mathcal{L}$ under the scheme described above that is history-based can be simulated by a stack algorithm which is $\mathcal{L}$-compatible.*

Proof. Regardless of the way the vector $\mathbf{y}$ is increased in step (ii), some (unique) primal variable x_k which corresponds to a dual constraint that was just made tight, will enter the solution in step (iii). For an index i, let $D_i = \{j : a_{ij} > 0\}$. Since we assume that the stack algorithm is compatible, the indices in D_i correspond to information (i.e. a_{ij}) that is revealed about item i. We consider the slack of dual constraint i: $c_i - \sum_{j\in D_i} a_{ij} y_j$. Now suppose the rate of increase of (unfrozen) dual variable y_j is r_j. Then the "time " to reach tightness in constraint i is

$$f(i) = \frac{c_i - \sum_{j\in D_i} a_{ij} y_j}{\sum_{\{j\in D_i \text{ and j is unfrozen}\}} a_{ij} r_j}.$$

We recall, however, that the rates r_j are not fixed throughout the runtime of the algorithm, and in fact we have rates $r_j^{(\ell)}$ that correspond to the rate of increase of the j'th dual variable in iteration ℓ. Consider the first ordering function. None of the dual variables are frozen and $\mathbf{y} = 0$, and therefore $f_1(i) = \frac{c_i - \sum_{j\in D_i} a_{ij} y_j}{\sum_{j\in D_i} a_{ij} r_j^{(1)}} = \frac{c_i}{\sum_{j\in D_i} a_{ij} r_j^{(1)}}$ is a function that can be computed given the information in each item. Notice that we justify the above statement by the fact that each $r_j^{(1)}$ does not depend on the input. Let k_1 be the index of the first item according to f_1, and let $t_1 = f(k_1)$. Notice that t_1 is precisely the "time" that is increased in the first iteration of the primal dual algorithm on this input. At this point the algorithm knows the identity of item k_1, and can deduce the frozen dual variables in the primal-dual algorithm it is simulating: these correspond to the constraints in which x_{k_1} is involved. Let's call this set F_1. (For convenience we also let $F_0 = \emptyset$.) It can be verified that the item next chosen by the primal-dual algorithm is the one that minimizes

$$f_2(i) = \frac{c_i - \sum_{j \in D_i} a_{ij} y_j}{\sum_{j \in D_i, j \notin F_1} a_{ij} r_j^{(2)}} = \frac{c_i - \sum_{j \in D_i} a_{ij} t_1 r_j^{(1)}}{\sum_{j \in D_i, j \notin F_1} a_{ij} r_j^{(2)}}.$$

Notice that since t_1 and F_1 are known to the stack algorithm, and since c_i and a_{ij} are available for item i, f_2 is a valid ordering function. Let k_2 be the item which attains the minimum of f_2. We now define $t_2 = f(k_2)$ and F_2 to be the constraints j in which x_{k_2} is involved. Recursively, we can show how to define f_s, x_{k_s}, t_s, F_s in the s th round by the corresponding values for previous rounds. Here is how.

$$f_s(i) = \frac{c_i - \sum_{j \in D_i} a_{ij} y_j}{\sum_{j \in D_i, j \notin F_{s-1}} a_{ij} r_j^{(s)}} = \frac{c_i - \sum_{j \in D_i} a_{ij} \sum_{0<\ell<s: j \notin F_{l-1}} r_j^{(\ell)} t_l}{\sum_{j \in D_i, j \notin F_{s-1}} a_{ij} r_j^{(s)}},$$

where we used the identity $y_j = \sum_{0<\ell<s: j \notin F_{l-1}} r_j^{(\ell)} t_l$.

Using the functions f_ℓ above, at every iteration the stack algorithm pushes the first item (i.e. the item i not currently in the partial solution which minimizes the value of $f_\ell(i)$) and continues. The pop phase is clearly a reflection of step (v) in the primal dual schema.

□

Remark 4. It is useful to view the above discussion in the context of the "standard" greedy algorithm for weighted Set-Cover, namely the algorithm that selects the next set S so as to minimize the weight of S divided by the number of uncovered elements in S. Note that commonly this is not considered to be an instantiation of a Primal Dual algorithm. Indeed this standard greedy algorithm does not fit the Primal Dual schema just outlined. However, a small extension to the schema will be sufficient to include this greedy algorithm. Namely, we extend the primal-dual framework so that any frozen dual variable can be removed from any inequality. With all frozen variables removed, it is simple to see that we obtain the standard greedy algorithm when we increase the value of uncovered elements uniformly. That is, when we ignore the frozen dual variables (i.e., the covered elements) from all inequalities, at any point of time the next dual constraint to become tight corresponds to the set with the most uncovered elements relative to its weight. A simulation of this richer schema by the Stack Model is still possible as can be seen by observing that the only change required in the simulation is simply that the expression $c_i - \sum_{j \in D_i} a_{ij} t_1 r_j^{(1)}$ should be replaced by $c_i - \sum_{j \in D_i, j \notin F} a_{ij} t_1 r_j^{(1)}$ where F is the current set of frozen dual variables, noting that F can be computed by the current information available.

The Importance of the Specific LP relaxation: It is important to note that *any* formulation of a covering or packing problem as an LP with positive coefficients can lead to a primal dual algorithm, which in turn can provide an ordering of the items. The stack model abstraction helps us to understand the power of history based primal dual algorithms under natural (albeit restricted) methods for the order in which the primal variables are taken into the solution. In order to show an inapproximability result for a suggested LP relaxation, we first have to provide an appropriate input-representation (for the items corresponding to the primal variables) and then prove a stack model inapproximation bound. Clearly, defining the type of information an item in the stack model holds is critical, since this determines the types of orderings that are possible in the execution of the algorithm. For example, in the vertex cover problem, in addition to the standard inequalities,

```
Algorithm $LRcov(\mathcal{F}, \mathbf{z}, \mathbf{w})$
    If $\emptyset$ is a solution for $(\mathcal{F}, \mathbf{z})$ then return $\emptyset$
    Construct a $\mathbf{w}$-tight weight function $\delta$ which is $r$-effective with respect to $(\mathcal{F}, \mathbf{z})$
    Let $j$ be the index not in the partial solution $\mathbf{z}$ with $\delta_j = \mathbf{w}_j$.
    $\mathbf{z}' := \mathbf{z} \cup \{j\}$
    $\mathbf{x} = LRcov(\mathcal{F}, \mathbf{z}', \mathbf{w} - \delta)$
    If $\mathbf{x} \cup \mathbf{z}$ is not a solution of $(\mathcal{F}, \mathbf{z})$, then $\mathbf{x}_j = 1$
    Return $\mathbf{x}$
End $LRcov$
```

Figure 2: The form of a local ratio algorithm for a covering problem [17].

we may look at the IP/LP relaxation that also includes inequalities forcing vertices in an odd cycle of size l to sum up to $(l+1)/2$. Notice that this type of tightening can be be derived from Chvátal-Gomory inequalites (see [58] for a discussion of Chvátal-Gomory inequalities) and also as a result of lift and project constructions (as discussed in the context of a hardness result in [8, 9]). If we manage to get a lower bound for an input representation in which a vertex knows all its adjacent vertices *and* all the odd cycles containing it, this would imply a lower bound for a primal dual algorithm applied to this relaxation.

The relation to local-ratio algorithms: Bar-Yehuda and Rawitz [17] present general primal dual and local ratio schemas for covering (see figure 2) and packing algorithms and establish an equivalence for such algorithms. A covering problem is specified by a set of constraints $\mathcal{F}$ and a weight function $\mathbf{w}$. Given a partial solution $\mathbf{z}$ (thought of as a subset of $\{1, 2 \ldots n\}$ or as a vector in $\{0,1\}^n$), the schema in [17] is recursively solved by an algorithm $LRcov(\mathcal{F}, \mathbf{z}, \mathbf{w})$ where the partial solution $\mathbf{z}$ is interpreted as enforcing the additional constraint that $\mathbf{z} \cap \mathbf{x} = \emptyset$ so that the solution $\mathbf{x}$ of $LR(\mathcal{F}, \mathbf{z}, \mathbf{w})$ extends $\mathbf{z}$ to a solution $\mathbf{z} \cup \mathbf{x}$ of $\mathcal{F}$. In order to obtain an r-approximation, the schemas in [17] are formulated in terms of a weight function δ which is *r-effective* with respect to $(\mathcal{F}, z)$ (when such a function exists). We omit the definition of such a weight function and refer to reader to [17]. By imposing an analogous history based restriction that the weight function δ is defined as a function of the information relating to the variables already set (or discarded) in previous iterations, we can simulate any such local-ratio algorithm in terms of a compatible stack algorithm (as described for history based primal dual algorithms). Minimality is ensured in the test that $\mathbf{x} \cup \mathbf{z}$ is not a solution of $(\mathcal{F}, \mathbf{z})$. The desired solution is obtained by calling $LRcov(\mathcal{F}, \emptyset, \mathbf{w})$. Any such recursive algorithm can obviously be expressed as an iterative algorithm using a stack.

2.4 Stack Model for Packing Problems

Following the discussion in Bar Noy et al [11], we now consider the formulation of the primal dual and local ratio schemas for packing problems in terms of a corresponding stack model. In fact, this framework considers the larger class of maximization problems where the subset of any feasible set is feasible with at most the same profit which, for example, includes the "cover by many" formulation of the cell selection problem in [5]. We consider the following primal dual framework similar to the one outlined for covering problems:

(i) Start with the (non-feasible) dual solution $\mathbf{y} = 0$, and an integral but feasible solution

$\mathbf{x} = 0$; set $\ell := 1$.

(ii) Continuously increase those dual variables y_j that occur in an unsatisfied dual constraint D_ℓ at rates r_j^ℓ until this dual constraint becomes satisfied; in doing so, other constraints may also get satisfied[3]

(iii) When the dual constraint becomes satisfied, the corresponding primal variable enters the solution;

(iv) As long as there are unsatisfied dual constraints, increment ℓ and go back to (ii).

(v) Go over the primal variables in the solution (i.e. those that were assigned value 1) in reverse order accepting variables that do not violate the primal packing requirements.

As in the stack model for covering problems, we have to formalize how the violated dual constraint is chosen. We omit the analogous details and simply describe the priority and stack models for packing problems. Namely, for priority algorithms we have a single phase and for stack algorithms we have a push phase and a pop phase. The single phase for priority algorithms (respectively, the push phase for stack algorithms), proceeds by using the same priority ordering mechanism (as in the model for covering problems) for determining the order in which consider items. Upon considering an item, the algorithm will either accept or reject the item for priority algorithms (respectively, push on the stack or reject for stack algorithms). Priority algorithms must maintain a feasible solution in all iterations (i.e. cannot accept an item if it will make the accepted set infeasible). In contrast, stack algorithms can allow an infeasible set of items on the stack and then use the pop phase to insure feasibility as follows. The pop phase starts with the empty set as its existing solution, and then adds a popped item to the existing solution unless the item renders the current solution infeasible. Whereas, priority algorithms use and provably need adaptive orderings for certain packing problems (see, for example, [23]), to the best of our knowledge, the known primal dual/local ratio algorithms that fit within the stack model are *fixed order* stack algorithms. Pseudo code for an adaptive (order) stack packing algorithm is presented in figure 3.

2.5 The Adversarial Approach

We briefly indicate how we can derive negative results for approximation ratios in the (adaptive ordering) stack model. An adversary initially presents a large set of potential input items. As the computation proceeds, the adversary is allowed to eliminate unseen input items (as long as a valid input instance can still be constructed). We note that the adversary is allowed this license since the ordering functions depend only on the item and any previously considered items. Eventually the algorithm will consider all input items that have not been eliminated and that will end the push phase. Then since the pop phase is completely determined, we can calculate the cost/profit of the stack algorithm and compare it to an optimal solution for the actual set of items (i.e. the input items not eliminated by the adversary and hence considered by the algorithm). For the case of fixed-order stack algorithms, the adversary can often construct the initial set of potential items in such a way that some "difficult structure" exists within a subset of the items no matter how the algorithm chose its order.

[3]In the fixed order algorithm of [11], the primal variables/dual constraints correspond to intervals that are being considered in the order of non-decreasing end-points. Satisfying a dual constraint corresponding to an interval will effectively decrease the residual profits of other intervals and hence cause their corresponding dual constraints to be satisfied. As in the Williamson [61] covering examples, all relevant dual variables are raised at a uniform rate.

```
Let $\mathcal{I}' \subset \mathcal{I}$ be the actual input set
While $\mathcal{I}'$ is not empty
    Determine a total ordering $\Pi$ on $\mathcal{I}$, the set of all possible remaining input items
    $next$ := index of input item $I$ in $\mathcal{I}'$ that comes first in the ordering
    As a function of all input items and decisions in previous iterations, make a decision
      as to whether to reject $I_{next}$ or push $I_{next}$ onto the stack
    Remove $I_{next}$ from $\mathcal{I}'$ and remove $I_{next}$ and all input items that proceed $I_{next}$ in $\Pi$ from $\mathcal{I}$
End While
$Sol := \emptyset$
While the stack is not empty
    $next$ := index of input item $I$ on top of stack
    Pop $I_{next}$ from stack
    Reject $I_{next}$ if $Sol \cup I_{next}$ is not feasible; else $Sol := Sol \cup \{I_{next}\}$
End While
```

Figure 3: The form of an adaptive stack algorithm for a packing problem.

Remark 5. The reader may benefit by noticing that there are three terms whose English meanings sound pretty similar, but that have very distinctive roles in this paper: rejection is the action of the algorithm when it decides decides not to push an item into the stack; deletion is the removal of an item in the pop phase; last elimination is what the adversary does when it decides that an item that has not yet been witnessed by the algorithm actually does not appear in the input.

2.6 Other primal dual algorithms

While many of the known applications of primal dual fall under "primal dual with reverse delete", there are several applications that do not fit within the stack model framework. Of these, one of the most prominent examples is Jain and Vazirani's [47] use of the primal dual method for the approximation of metric facility location and k-median problems [47]. Notice that these problems are not covering or packing problems (specifically for facility-location we think of the usual representation in which indicator variables are used for facility-openings and for city-facility connections), and therefore may inherently be in need of a more specialized algorithmic paradigm within the primal dual context [4]. A more basic example is the Steiner tree algorithm in [53] for the special class of quasi-bipartite graphs. Their method provides a $3/2 + \epsilon$ approximation by combining local search with primal dual and using the so called *bidirected* LP relaxation as a starting point. (This method utilizes a non-standard input representation by considering each edge as two directed edges and hence automatically falls outside of our model.) Another example of a primal dual/local ratio algorithm that does not fit our model are the so called *fractional* local ratio/primal dual algorithms ([15],[18]) which first solve an LP, and then use the LP solution as a guide to ordering items.

[4]The second phase of the Jain and Vazirani [47] primal dual algorithm uses a more involved process to delete certain facilities that were opened during the first phase.

3 The Set Cover Problem

We consider the classic minimum set cover problem. We are given a family of subsets $S_1, S_2, \ldots, S_m$ of a ground set U of size n. In addition, each set S_i has a cost c_i. The goal is to minimize the cost of a collection of the S_i that cover every element in U. The unweighted version of the problem seeks to minimize the number of subsets chosen.

The well known natural greedy algorithm for set cover (see the presentation in Vazirani [60] and the tight analysis in Slavík [59]) selects sets with minimum cost per size ratio, and continues recursively on the remaining set of uncovered items with the remaining sets. The natural greedy algorithm yields an $H(n)$ approximation. Notice that this algorithm is a priority algorithm, and hence a stack algorithm in which no item is rejected during the push phase and nothing is deleted in the reverse-delete stage.

Raz and Safra [54] showed that if $P \neq NP$, then for some constant $c > 0$, set cover cannot be approximated within a ratio of $c \log n$ in polynomial time. Using a stronger complexity assumption, namely that $NP \not\subset DTIME(n^{\log \log n})$, Feige [39] was able to show an almost tight bound $(1-\epsilon)H(n)$ for all $\epsilon > 0$. But these hardness results do not apply to our model as the stack algorithm is not restricted to run in polynomial time (i.e. in deciding what item to consider next and whether or not to push the item onto the stack). Furthermore, stack algorithms may be nonuniform, in the sense of allowing a different algorithm for each n, the number of input items.

As a warmup, we show how to get a $(\log n+1)/2$ inapproximability for the weaker priority algorithm model. The following is an algebraic reformulation of a similar result from [6].

Let the ground set be the discrete cube $\{0,1\}^\ell$ where $\ell = \log n$, and let the initial input be the 2ℓ sets corresponding to the facets of the cube, in other words, the sets $\{x \in \{0,1\}^\ell : x_i = b\}$ for each $i = 1, \ldots, \ell$ and each $b \in \{0,1\}$. Notice that this input consists of ℓ pairs of complementary sets. Assume without loss of generality that the first set chosen is $\{x : x_1 = 0\}$. We argue that it must be taken, as otherwise the adversary may announce that the opposite face, $\{x : x_1 = 1\}$, is the only remaining set in the input, which means that while there is a covering, the algorithm would not produce one. The algorithm therefore accepts $\{x : x_1 = 0\}$. In response, the adversary eliminates the set $\{x : x_1 = 1\}$. At this point we remain with the exact same problem, but with ℓ decreased by 1. Since for $\ell = 1$ there are 2 sets that need to be accepted, we get that there are $\ell + 1$ sets that the priority algorithm takes instead of just two. This gives the desired bound.

The above motivates the additional power of stack algorithms : if the last two sets we take are a complementary pair, then the recursive nature of the algorithm means that all the sets but this pair will be eliminated in the pop phase. The new set up we define for the lower bound is therefore more involved, but still uses a succinct algebraic description. We note that a somewhat similar construction is used in [60] to show that the integrality gap of the natural LP relaxation for set cover is $\Omega(\log n)$.

As before, the ground set U is the cube $\{0,1\}^{\log n}$. The initial input to the algorithm consists of $2(n-1)$ sets, S_v^b where $v \in \{0,1\}^{\log n} \setminus \{0\}^{\log n}$, and $b \in \{0,1\}$. Set S_v^b contains all points x for which $\langle v, x \rangle = b$ where $\langle \cdot, \cdot \rangle$ is the inner product over $\mathbb{GF}_2$. (In what follows all algebraic operations are over $\mathbb{GF}_2$.)

The sets can be viewed as the inverse images of 0 (1) of all the nontrivial Fourier characters of the boolean cube. We note that for any v, S_v^0 and S_v^1 are disjoint sets and that $S_v^0 \cup S_v^1 = U$.

We require a simple lemma relating the combinatorial properties of sets in our system to the algebraic properties of the vectors that define them:

Lemma 6. *For any set of linearly independent vectors* $\{v_1, \ldots, v_k\}$ *and any choice of* $b_1, \ldots, b_k$*, the number of elements of* U *left uncovered after* $\{S_{v_1}^{b_1}, \ldots, S_{v_k}^{b_k}\}$ *are selected is exactly* $n/2^k$*. In particular, any family of sets whose corresponding vectors are linearly independent does not cover* U *completely.*

Proof. The elements covered by the above sets are $S_{v_1}^{b_1} \cup \ldots \cup S_{v_k}^{b_k}$, hence the uncovered ones are

$$S_{v_1}^{1-b_1} \cap \ldots \cap S_{v_k}^{1-b_k}.$$

Therefore, x is uncovered if and only if it is a solution of the linear system of equation

$$\langle v_1, x\rangle = 1 - b_1, \ldots \langle v_k, x\rangle = 1 - b_k.$$

Since the v_i are linearly independent, the dimension of the solution space of this system is $l - k$ and so it contains $2^{l-k} = 2^l/2^k = n/2^k$ elements and the first part of the lemma follows. The second part is obvious (in the extreme there are ℓ such independent vectors in which case there is one uncovered element). □

Using the set-system above, we will show that no stack algorithm can achieve a set cover smaller than $\log_2(n+1)$, while the optimal set cover is of size 2. At stage i of the stack algorithm, the algorithm chooses a set $S_{v_i}^{b_i}$ to look at next. Recall that the adversary may eliminate sets from the input as the algorithm proceeds.

At each step i, $i < \log n$, suppose the algorithm accepts $S_{v_i}^{b_i}$. Then, the adversary eliminates all the sets S_v^b with $v \in \text{span}\{v_1, \ldots, v_i\}$. Notice that this strategy ensures that as long as $i \leq \log n$, the set of vectors defining the sets in the stack are linearly independent. In particular, for any vector v, S_v^0 and S_v^1 cannot both be on the stack.

In the other case in which the algorithm rejects the set $S_{v_i}^{b_i}$, we argue that the algorithm may fail to generate *any* set cover. Indeed, as a response the adversary simply eliminates all unseen inputs except for $S_{v_i}^{1-b_i}$. Note that $v_i \notin \text{span}\{v_1, \ldots, v_{i-1}\}$ so this is a valid strategy. But even if the algorithm takes that last set, the stack still contains sets with corresponding linearly independent vectors $\{v_1, \ldots, v_i\}$, which by Lemma 6 does not form a cover. But of course one exists as the input contains the two complementary sets $\{S_{v_i}^{b_0}, S_{v_i}^{b_1}\}$. This argument holds also for $i = \log n$.

Now, assuming that the algorithm continues to accept, after the $(\log n)$-th step, the algorithm has accepted $\log n$ sets whose defining vectors are linearly independent, leaving exactly $n/2^{\log n} = 1$ uncovered element. The adversary will eliminate all sets *except* for $S_{v^*}^0$ and $S_{v^*}^1$, where

$$v^* = \sum_{i=1}^{\log n} v_i.$$

It is important to observe that these sets were not eliminated before as $v^* \notin \text{span}\{v_1, \ldots, v_{\log n - 1}\}$; indeed, if it were then $v_{\log n} = v^* - \sum_{i=1}^{\log n - 1} v_i$ would also be in that span, contradicting the fact that it was chosen in the $(\log n)$-th round. At this point our the "game" is over, and the input to the algorithm is determined. The stack contains $\log n$ linearly independent sets and there are the two remaining sets that the algorithm must consider. Clearly, the sets $S_{v^*}^0$ and $S_{v^*}^1$ constitute a set cover of size 2. It remains to argue that the algorithm must take $\log n + 1$ sets. Since only one element is left uncovered before the two complementary sets are considered, one of the sets is contained in the union of the sets on the stack. This means that this set will be deleted in the pop phase and we may as well assume the algorithm rejects it. The algorithm now has $\log n + 1$ sets on the stack.

We claim that all of these sets must survive the pop phase. Indeed, by the special choice of v^* the vectors $v_1, \ldots v_{\log n}, v^*$ are in general position, that is no $\log n$ of them are linearly dependent. Since, by the independence of $v_1, \ldots v_{\log n}$ there may be only one dependency between the $\log n+1$ vectors, and since summing *all* of them is a dependency, we know that there are no other dependencies, and so the vectors are indeed in general position. We use Lemma 6 again to deduce that no strict sub-family of sets may cover the ground set, and in particular no set can be deleted in the pop phase. We have established

Theorem 7. *No stack algorithm can achieve an approximation ratio better than* $\log(n+1)/2$ *for the unweighted minimum set cover problem, where* n *is the number of elements in the ground set to be covered.*

4 The Steiner Tree Problem

In the Steiner Tree problem we are given an edge weighted graph $G = (V, E, w)$ and a set of distinguished *terminal* vertices T. The goal is to choose a subset of edges that connect all terminal vertices, so as to minimize the sum of the weights. The primal-dual algorithm that achieves a 2-approximation uses the natural LP relaxation in which the IP constraints dictate that every cut separating T is crossed by an edge. We sketch how a local-ratio algorithm is used to get a 2-approximation [17]. Pick an edge e for which the value of $\{w_e/|e \cap T| : |e \cap T| \neq \emptyset\}$ is minimized, and recursively solve the problem in which e is contracted to a vertex, and the revised weight of an edge f is now $w(f) - w(e) \cdot \frac{|f \cap T|}{|e \cap T|}$. Notice that if edge e is appended to a solution from the recursive call, a solution to the original problem is obtained. The reverse delete step allows the algorithm to avoid adding e if it is not necessary. Interestingly, this step can be shown to be essential, as the analysis depends on the fact that the resulting set is minimal with respect to set inclusion. Moving to the stack model abstraction for the problem, we consider two possible input representations. In each representation, the items are the edges. For the first *standard edge input model*, an edge is represented by its cost, and the identity of its endpoint vertices. Further, we assume the algorithm knows (as global information) the names of the vertices and the set of terminals. In the second *complete edge input model*, we assume additionally that the entire graph structure is known and that an edge is simply represented by its cost, as all other information is known globally. In the standard edge model, the adversary will set up an initial graph, but may eliminate unseen edges over the course of the algorithm's execution.

What is known about the Steiner Tree problem within and outside of the scope of our stack model? Unlike the set cover problem, it is still plausible (relative to known complexity based hardness results) that there is a conceptually simple and efficient approximation algorithm that improves upon currently known polynomial time approximation algorithms. Chlebík and Chlebíková [30] show that it is NP-hard to obtain an approximation ratio better than $\frac{96}{95} \approx 1.01$. The best polynomial time approximation algorithm for the Steiner Tree problem, due to Robins and Zelikovsky [55], achieves approximation ratio 1.55. For the class of quasi-bipartite graphs (i.e. graphs not containing edges between Steiner nodes), the Robins and Zelikovsky algorithm achieves a ratio ≈ 1.28 and the NP-hardness bound is $\frac{128}{127} \approx 1.008$ [30]. Like almost all recent Steiner Tree algorithms (see, in particular, Chakrabarty et al [27] and Könemann, et al [50]), the Robins and Zelikovsky algorithm begins by computing the metric completion of the input graph (i.e. by adding missing edges with cost equal to the least cost path in the graph). Hence, perhaps the most relevant inapproximation result we obtain for the Steiner Tree problem is the $\frac{7}{6}$ bound with respect to the complete edge model derived in section 4.2. Closely related to this inapproximation is the $\frac{73}{60}$ approximation by Gröpl et al [44] for uniform quasi bipartite graphs where uniform means that all edges incident on the Steiner nodes have the same cost. This algorithm has a greedy aspect to it but falls outside the stack model. More specifically, the Gröopl et al [44] algorithm initially computes the full components (which can be easily done within the complete edge model for uniform quasi bipartite graphs) and also the costs of these full components but then greedily pieces these full components together (in a manner not consistent with the pop stage of a stack algorithm). The standard edge model is more appropriate if we do not assume an initial metric completion (e.g. if the goal is a near linear time approximation algorithm). We note that our inapproximation results apply to bipartite graphs for both input models. Recently, Kwon and Edmonds [36] have improved our $\frac{4}{3}$ lower bound for the standard edge model to achieve a $2 - O(\frac{1}{|V|})$ inapproximation and moreover have shown that a 2-approximation can be achieved for the Steiner Tree problem on arbitrary graphs in the standard edge model by an efficient stack algorithm, matching the near linear time bound of Melhorn [52]. Most recently, substantial progress has been made on the Steiner tree problem for both uniform quasi bipartite graphs (Chakrabarty, Könemann and David Pritchard [28]) and general graphs (Byrka, Grandoni, Rothvoß and Laura Sanità [25]). Specifically, amongst other results, Chakrabarty et al obtain a 1.216 approximation for uniform quasi biparitite graphs and Bryka et al obtain a $\ln 4 + \epsilon < 1.39$ approximation for general graphs thus improving upon the 1.55 approximation of Robins and Zelikovsky [55].

Könemann, Pritchard and Tan [50] refer to the Robins and Zelikovksy algorithm as a greedy algorithm and provide an interpretation of the Robins and Zelikovksy algorithm as a primal dual algorithm. The greedy aspect of the algorithm can be explained as follows. Initially, a Steiner tree solution is obtained by computing the minimum spanning tree of the subgraph induced by the terminal nodes. The algorithm will iteratively improve the current solution by irrevocably adding Steiner nodes and recomputing a minimum spanning tree on the current set of nodes (and in doing so, permanently deleting some previously selected edges). The precise manner in which the Steiner nodes are chosen depends upon a priority function of "full components" (whose Steiner nodes will be added) relative to the current Steiner solution. We claim that the choice of the next full component to be added as well as the computing of full components and the iterative computing of the minimum spanning trees can all be done within the push stage of a stack algorithm (given that the complete graph structure is known and that edges are the input items represented by the end points and their costs). However, this algorithm is beyond the scope the stack model (and our formalization of the schema in Williamson [61]). Indeed, the removal of edges is not performed in the Last-In-First-Out order used by stack algorithms. Rather, more costly edges (now causing cycles) are removed to maintain minimality. The model being suggested then is a priority model for covering problems that allows revocable acceptances subject to the requirement that the partial solutions being maintained can always be extended to minimal solutions. Similarly, the Chakrabarty, Devanur and Vazirani [27] strongly polynomial $\sqrt{2}$-approximation for quasi bipartite graphs iteratively recomputes minimum spanning trees, irrevocably adding new Steiner nodes. The $\frac{4}{3}$-approximation in the same paper moves further from the schemas we have been discussing by potentially removing Steiner nodes.

We will present lower bounds for both stack input models motivated by the priority lower bound of Davis and Impagliazzo [33]. For both input models we will consider bipartite graphs $G =< T \cup S, E >$, where $E \subseteq T \times S$, T is the set of terminal nodes and S is the set of Steiner nodes. (Obviously these nemesis graphs are also quasi-bipartite.) We make the following simple observation for such bipartite graphs:

Observation 8. *Any Steiner tree for $G =< T \cup S, T \times S >$ must contain at least one Steiner node s with degree greater than one.*

This is simply because the terminal nodes are not connected by themselves, and so Steiner nodes must be used. Obviously a Steiner node that is used cannot be a leaf. The observation implies that in any stack algorithm, at least two edges adjacent to some Steiner node are pushed onto the stack. The next lemma tells us that the first such pair will also survive the pop phase.

Lemma 9. *Consider the first time that the stack algorithm pushes two edges adjacent to some Steiner node. Then the solution produced by the algorithm must contain these two edges.*

Proof. Let e, f be these edges, and assume e is deleted in the pop phase. This means that when e was popped there was a cycle C in the union of the edges in the current partial solution and the edges currently on the stack, rendering e redundant. Moreover, among the edges of C, e is the last to be pushed since otherwise the last pushed edge in C would have been deleted before considering e. Since the graph is bipartite, any cycle must also pass through a second Steiner node, and must therefore use two of the edges (say g, h) out of that second Steiner node. Since e was the last edge in C to be pushed, g and h were pushed before e, contradicting the fact that e, f are the first pair of edges connected to the same Steiner node to be pushed. □

4.1 A lower bound for stack algorithms using the standard edge input model

Theorem 10. *For the standard edge input model, no stack algorithm can achieve a worst-case approximation ratio better than $4/3$ for the (unweighted) Steiner tree problem. This inapproximation result holds for bipartite graphs.*

Proof. The initial input graph is $G =< T \cup S, T \times S >$, with $T = \{t_1, t_2, t_3\}$ and $S = \{s_1, s_2\}$. Let us first review the possible solutions and their costs. The stars rooted at s_1 and s_2 both have cost 3, while the other possible Steiner trees must use both Steiner nodes and have cost 4.

Using Observation 8, we consider the first time the second edge connected to any Steiner node (say to s_1 without loss of generality) is pushed onto the stack. If the third edge connected to s_1 was not already rejected, the adversary eliminates it. We now claim that the algorithm can no longer achieve a solution with cost 3. Indeed, by Lemma 9 it will produce a solution with at least two edges connected to s_1, but since the third edge was either rejected (by the algorithm) or eliminated (by the adversary), none of the stars rooted at Steiner nodes are possible solutions. On the other hand the optimal solution is the star rooted at s_2. This leads to an approximation lower bound of 4/3. □

4.2 A lower bound for stack algorithms using the complete edge input model

We next show that there is a nontrivial approximation lower bound even when the algorithm is allowed to know in advance the edges of the graph. To show that we consider the complete bipartite graph $G =< T \cup S, T \times S >$, where $T = \{t_1, t_2, t_3, t_4\}$ are the terminals and $S = \{s_1, s_2, s_3\}$ are the Steiner nodes. Furthermore, all edge costs are in $\{1, 2\}$. Therefore, the only thing not known in advance is whether an edge has unit cost or not. This state of affairs relates quite nicely to the discussion about what LP relaxations we may consider: when all the information is known but the cost vector, then *any* LP relaxation with the natural variables and with positive constraint coefficients is captured by the model.

Initially, the algorithm is allowed to "choose" the cost that it prefers for each edge it considers; more precisely, it will be given the edge cost (i.e. cost 1 or cost 2) that it has currently given highest priority to in its ordering among all <edge,cost> pairs. After each stage, the adversary has the option of fixing the costs of any edges that have not yet been seen. Note that this will be consistent with the ordering chosen by the algorithm, since for any unseen edge, both possible edges (i.e. the one with cost 1 and the one with cost 2) must have had lower priority at every stage than the edges that the algorithm actually considered. The adversary will set edge costs so as to make the unique optimal Steiner tree be a star rooted at some Steiner node while the algorithm will have already made decisions that prevent him from choosing this tree.

A useful observation is that the number of edges of a Steiner tree in the above graph will be 4, 5 or 6 depending on whether the number of Steiner nodes participating are 1,2 or 3 respectively.

Lemma 11. *If the algorithm rejects an edge before or at the first time two edges connected to the same Steiner node were seen, then it cannot achieve a better approximation ratio than* $7/6$.

Proof. Let e be the rejected edge, and let s be its Steiner node. The adversary then sets to 1 the cost of all unseen edges connected to s, and all other unseen edges are set to have cost 2. We now argue that the unique best solution is the star rooted at s. This will show the lemma, as the algorithm can no longer pick that star. To see this notice that this star has cost at most $2 + 2 + 1 + 1 = 6$. The stars rooted at other Steiner nodes have at most one edge that was seen before the adversary intervened, hence at least 3 edges with cost 2 and so their cost is at least $2 \times 3 + 1 = 7$. A solution which is not a star but which contains 3 edges connected to the same Steiner node must have at least 2 of these edge of cost 2, and a total of 5 edges, hence will have cost at least $2 \times 2 + 3 \times 1 = 7$. Otherwise, a solution of 6 edges is used, and obviously not all of them may have unit cost, which again implies a cost of at least 7. □

Lemma 12. *If at least two of the first three edges considered are adjacent to the same Steiner node* s*, and both edges are pushed, then the algorithm cannot achieve an approximation ratio better than* $5/4$.

Proof. Lemma 9 guarantees that the solution produced by the algorithm must contain the first two pushed edges adjacent to s. The adversary sets the unseen edges adjacent to s to have cost 2 and all other unseen edges to have cost 1. The optimal solution is a star rooted at a Steiner node $s' \neq s$ and has cost 4 while the algorithm's solution must either be a star rooted at s (having cost at least 6) or must contain at least two Steiner nodes (and therefore at least 5 edges) forcing the cost to be at least 5. □

We are now ready to state the theorem for this input model.

Theorem 13. *For the complete edge input model, no stack algorithm can achieve a worst-case approximation ratio better than 7/6 for the Steiner tree problem (even if all edge costs are in $\{1, 2\}$). This inapproximation result holds for bipartite graphs.*

Proof. Let $< e_1, e_2, e_3 >$ be the first three edges considered by the algorithm (not necessarily in that order) during the push phase. In view of Lemmas 11 and 12, we fix our attention to the case where e_1, e_2, e_3 are pushed and touch all Steiner nodes. Note also that the adversary has not yet intervened. Assume e_i is adjacent to node s_i. We consider two cases:

Case 1: The edge costs (as a multiset) for $\{e_1, e_2, e_3\}$ are not $\{1, 2, 2\}$. Let us say that e_4 is the next edge to be considered by the algorithm and that e_4 is adjacent to edge e_1 (and hence to vertex s_1). We denote by d the cost of e_1. Without loss of generality, e_2 would have cost not bigger than the cost of e_3. Notice that since the costs of $\{e_1, e_2, e_3\}$ are not $\{1, 2, 2\}$ this also means that the cost of e_2 is at most d. Using Lemma 11, we can assume that the algorithm has pushed e_4. The adversary now sets all unseen edges adjacent to s_1 to have cost 2 while all other unseen edges have cost 1. The optimal solution is the star rooted at s_2 having cost at most $3 + d$. By Lemma 9, the algorithm must take e_1 and e_4 and therefore have a cost at least $4 + d$ since either its solution is the star rooted at s_1 (for a cost of at least $5 + d$) or the solution contains at least two Steiner nodes (and hence at least 5 edges including e_1) for a cost of at least $4 + d$.

Case 2: Say e_1 has cost 1 and both e_2 and e_3 have cost 2. The adversary sets all three unseen edges adjacent to s_1 to have cost 2. Let e_4 be the next edge that is pushed. We claim that no edge out of s_2 or s_3 was rejected prior to seeing e_4 as otherwise the relevant star can be made an unattainable unique optimal solution, using the argument in the proof of Lemma 11. Let s_i be the Steiner node adjacent to e_4. The adversary sets the cost of all unseen edges out of s_i to be of cost 2 and the remaining unseen edges are set to have cost 1. Notice that this means that there is a Steiner tree star rooted at a Steiner node $s \neq s_1, s_i$ with edge costs $\{2, 1, 1, 1\}$, and hence a solution of cost 5. But by Lemma 9, e_i and e_4 must both be included in the solution of the algorithm. Notice that by the strategy of the adversary and the $\{1, 2, 2\}$ assumption, the sum of costs of e_i and e_4 is at least 3. Considering the star at s_1 (or s_i), at least two edges have cost 2 which leads to a solution cost of at least 6. Otherwise, the solution includes at least two Steiner nodes (and hence 5 edges including the edges e_i and e_4) so that the algorithmic solution has cost at least 6. □

5 Packing Problems: Interval Scheduling and Bandwidth Allocation

Bar Noy et al [11] provided the first use of a local-ratio/primal dual approximation algorithm for a packing problem. Following [11], the local ratio method has been applied to various packing problems such as the rectangle alignment problem [20], and demand maximization in cellular networks [5]. A unification and extension of local ratio results for maximum independent set problems can be found [3] and [62]. We note that all these algorithms can be realized by *fixed order* stack algorithms. Several problems are discussed in [11], the most general one being the NP-hard bandwidth allocation problem (BAP). Here, each job is a set of intervals where an interval is represented by its starting time, its finishing time, its profit, its bandwidth and the job to which it belongs. The optimization problem is to schedule jobs so as to maximize the overall profit subject to the condition that at most one interval per job is scheduled and at any point of time t, the sum of bandwidths of intervals scheduled at time t will not exceed a given maximum capacity. Without loss of generality we may assume that the maximum capacity is 1 and that all bandwidths are at most 1. We shall restrict ourselves to the special case that there is only one interval per job. Even in this restricted case, the problem generalizes both the interval scheduling problem (on any number of machines) as well as the knapsack problem. Currently the best deterministic polynomial time approximation for BAP is the $\frac{1}{3}$ approximation of Calinescu et al [26] which combines a an optimal dynamic programming algorithm for "large bandwidth" intervals with a $\frac{1}{2}$-approximation for "small bandwidth" intervals.

We first consider the (polynomial time solvable) special case of weighted interval scheduling on m identical machines ($m - WISP$) where (in the terminology of the bandwidth problem) every interval has bandwidth equal to $\frac{1}{m}$. That is, the optimization problem is to schedule intervals on m machines so that any two intervals scheduled on the same machine cannot overlap. We show that 0.913 is an inapproximation bound[5] for the approximation of the $2 - WISP$ problem by any fixed order (packing) stack algorithm. This should be contrasted with the optimal local ratio algorithm for one machine interval scheduling ($1-WISP$) and the 2/3-approximation supplied for $2-WISP$ in [11]. We also note that an optimal algorithm for $m - WISP$ can be obtained by an $O(n^m)$ time dynamic programming algorithm or a time $O(n^2 \log n)$ min cost max flow based algorithm [7].

For the interval scheduling problem (respectively, the more general bandwidth allocation problem) the natural input representation is that the input items are intervals represented by their start times, their finish times and their profits. This representation is thus sufficient for the natural LP formulation in which constraints are associated with time points t and these constraints bound the number (or the bandwidth) of intervals that can be scheduled at time instance t. For our packing results, we are thus far only able to provide bounds for fixed order stack algorithms. This, however, does capture the one "meta-algorithm" that Bar Noy et al use to solve various cases of the bandwidth allocation problem. The fixed order there is determined by non-decreasing finishing times which is also the order used for the optimal greedy algorithm for unweighted interval scheduling, and for the one pass algorithms of Erlebach and Spieksma [37]. We emphasize that our bound does not require that this is the ordering used by the algorithm.

Theorem 14. *For interval scheduling on two machines, no fixed ordering stack algorithm can achieve a constant approximation factor better than* $\frac{\sqrt{30}}{6} \approx 0.913$.

We remark that Theorem 14 can be extended to any number m of machines, and leads to a $1 - O(1/m)$ inapproximability result which limits to 1 as m increases. To put this in perspective, this still leaves a considerable gap from the approximation ratio that is achieved by the local ratio algorithm [11], which $\frac{1}{2-1/m}$ for $m \geq 2$ which limits to 1/2 as m gets large.

Proof. To provide a bound for a fixed-order model (be it priority algorithm, backtracking algorithm [4], or a stack algorithm), it is often useful to provide an initial input set and claim that regardless of the ordering of elements in that set, some combinatorial property must apply to a subset of the initial set. This is analogous to Ramsey phenomena in graphs; i.e. in every coloring we can say something about one of the color classes. Restricted to such an ordered subset, which we call a *forbidden configuration*, we are able to bound the quality of the algorithm. Specifically, the adversary will eliminate all items but the ones participating in the configuration, whence an

[5]As this is a maximization problem we will present approximation ratios to be less than or equal to 1.

inapproximation bound for that configuration immediately implies a similar bound for the original problem. We describe the proof in a bottom up fashion, that is we start by first describing the *forbidden configurations* of the input that we will use.

Claim 15. *Consider the configuration FC1 in which intervals I, J, K appear in this order with $I \cap J \cap K \neq \emptyset$ and that have profits p_I, p_J, p_K respectively that satisfy $p_J < p_I, p_J < p_K$. Then for such a configuration (namely, when the input consists of the three intervals I, J, K, where I appears before J and J appears before K, or a subset of these three that is constent with that order) the best approximation ratio achievable is*

$$\max\{p_I/(p_I + p_J), (p_I + p_J)/(p_I + p_K), (p_J + p_K)/(p_I + p_K)\}. \tag{1}$$

Proof. To see the claim, consider the action of the algorithm on an input that contains a (not necessarily proper) subset of the intervals I, J, K in that order. If the algorithm decides to reject an interval, the adversary eliminates all remaining intervals. This leads to approximation ratios $0, p_I/(p_I + p_J)$ or $(p_I + p_J)/(p_I + p_K)$ when rejecting the first, second or third items respectively. In the more interesting case, all intervals are accepted to the stack, and in the pop phase I will be deleted since J and K are already in the solution. Therefore the algorithm achieves $p_J + p_K$ while $p_I + p_K$ is possible, and the bound (1) follows. □

Claim 16. *Consider the configuration FC2 in which intervals I_1, I_2, J, K have profits with profits p_I, p_I, p_J, p_K respectively. I_1 and I_2 are disjoint, and $I_i \cap J \cap K \neq \emptyset$ for $i = 1, 2$. Moreover, $p_I < p_J < 2p_I, p_J < p_K$ and the ordering is either I_1, I_2, J, K or K, J, I_1, I_2. Then for such a configuration (namely, an input consisting of at most these four intervals, and an ordering as described above) the approximation ratio is at least*

$$\max\{2p_I/(2p_I + p_J), (2p_I + p_J)/(2p_I + p_K), (p_J + p_K)/(2p_I + p_K)\}.$$

Proof. Suppose first that no interval gets rejected. Then either I_1 and I_2 get deleted in the pop phase (first ordering) or K does (second ordering). Hence instead of the optimal $2p_I + p_K$ the algorithm achieves $\max\{p_J + p_K, 2p_I + p_J\}$. Now suppose some intervals get rejected, and assume first that the ordering is I_1, I_2, J, K. If the adversary reacts to any rejection by eliminating all remaining intervals, this gives upper bound on the approximation of $0, p_I/2p_I, 2p_I/(2p_I+p_J), (2p_I+p_J)/(2p_I + p_K)$ when the first rejected interval is (respectively) I_1, I_2, J, K. For the other ordering, namely K, J, I_1, I_2, we get approximation ratios of $0, p_K/(p_K + p_J), (p_J + p_I)/(2p_I + p_K)$ when the first rejection is (respectively) K, J, I_2, using a similar argument as before. If I_1 is the first rejected interval then the adversary does not remove I_2 and instead of the optimal profit of $2p_I + p_K$ the algorithm either obtains profit $p_J + p_I$ by pushing I_2 or (the better) profit $p_J + p_K$ by rejecting I_2. To sum up, the configuration FC2 implies an upper bound on the approximation of

$$\max\{1/2, 2p_I/(2p_I + p_J), (2p_I + p_J)/(2p_I + p_K), p_K/(p_J + p_K), (p_J + p_K)/(2p_I + p_K)\}. \tag{2}$$

Noting that $1/2 < p_K/(p_J + p_K) < (p_J + p_K)/(2p_J + p_K) < (p_J + p_K)/(2p_I + p_K)$, this simplifies to

$$\max\{2p_I/(2p_I + p_J), (2p_I + p_J)/(2p_I + p_K), (p_J + p_K)/(2p_I + p_K)\}. \tag{3}$$

□

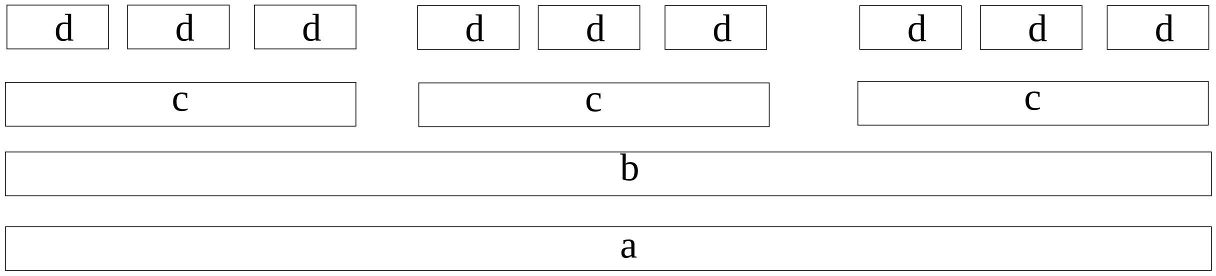

Figure 4: Initial input for interval scheduling on two machines used for Theorem 14 (with the corresponding profits)

Our goal now is to show the Ramsey-type phenomenon with respect to these two forbidden configurations; namely, construct a set of input intervals so that regardless of the ordering imposed by the algorithm, one of the two forbidden configurations can be obtained as a subsequence.

The initial input we take is a laminar set of intervals (i.e., every pair of intervals are disjoint, or one is contained in the other) described in Figure 4. We will fix the variables a, b, c, d that determine the profits of the intervals later on, and for now we only require that $a > b > c > d$, that $2c > a$ and that $2d > b$. To simplify the analysis (and without degrading the resulting bound), we will also let $a = 2d$.

We will abuse notation and sometimes just refer to an interval by its profit. We will show that for every ordering of these intervals, one of two cases must occur. In the first case, a c-interval or a d-interval occurs between two other intervals that contain it and have greater profit; for example, using $\prec$ to denote the ordering between intervals, consider the ordering $J \prec I \prec K$ where I is the leftmost d-interval, J is the leftmost c-interval and K is either the b-interval or the a-interval. This results in forbidden configuration FC1. In the second case, a b-interval occurs between an a-interval and two c-intervals that are contained within the b-interval, or a c-interval appears between a b-interval and two d-intervals that are contained in the c-interval. This results in forbidden configuration FC2. By appropriately substituting the a, b, c, d values for p_I, p_J, p_K in the bounds of Claims 15 and 16 and ignoring dominated terms (i.e. those that are guaranteed by the conditions on a, b, c, d not to be the maximum), we obtain from (1) and (3) the following bound on the approximation factor achieved by any algorithm that chooses an ordering containing one of these forbidden configurations:

$$\max\{(c + d)/(c + b), (a + c)/(a + b), (a + d)/(a + c), (2c + b)/(2c + a)\} \tag{4}$$

We now show that forbidden configurations are unavoidable. First, notice that both configurations are invariant with respect to inversion of the ordering. In other words, either ordering σ and σ^{-1} with both contain forbidden configurations, or neither will. We assume that forbidden configuration FC1 is avoided and show that in this case configuration FC2 must be present. If FC1 does not appear, then the a interval and the b interval must appear consecutively in the order. Of the three c-intervals, at least two will appear before or after the a and b intervals. In light of the comment above about inverting the order, we assume without loss of generality that two c-intervals appear *before* the a and b interval. We (i.e. the adversary) will then remove the other c-interval, and the d-intervals contained in it. Another condition implied by the absence of configuration FC1 is that each d-interval must appear before or after all of the intervals containing it. Consider the d intervals contained in the earliest c-interval Since the d-intervals appear first or last among the intervals we may assume without loss of generality that two of the aforementioned d-intervals appear before all, or after all of the two c-intervals and the a and b intervals. We remove the third d-interval. At this point, we know that there is an ordering of the remaining 6 intervals with profits $\{a, b, c, c, d, d\}$ where the ordering is either $d \prec d \prec c \prec c \prec a \prec b$ or $c \prec c \prec a \prec b \prec d \prec d$, or these orderings with a and b swapped. It now remains to notice that these four arrangements contain at least one forbidden FC2 configuration. In the first ordering we find the FC2 configuration $d \prec d \prec c \prec a$, and in the second ordering we either have $a \prec b \prec d \prec d$, if a appears before b or $c \prec c \prec b \prec a$, if b appears before a. We are left with setting values to a, b, c, d so as to satisfy the inequalities in the definitions of the forbidden configurations, and to minimize the maximum in (4). We minimize by setting $(a, b, c, d) = (10, 8, 3\sqrt{30} - 10, 5)$ which leads to a lower bound of $\frac{\sqrt{30}}{6} \approx 0.913....$

$\square$

Returning to the more general bandwidth allocation problem, as previously mentioned the local ratio method in [11] is a fixed ordering stack algorithm in which the ordering is determined by interval finishing times. In that paper, it is shown that the algorithm can achieve an approximation ratio of 1/2, when all bandwidths are restricted to be at most 1/2. We will now show that for the given ordering scheme no constant approximation ratio is possible when arbitrary bandwidths are allowed, and that the approximation factor of 1/2 is tight for the given restriction.

Theorem 17. *For any positive integer k, when all bandwidths are restricted to being at most $1/k$, and the ordering is the fixed ordering by finishing times, no stack algorithm can achieve an approximation ratio of $1 - 1/k + \epsilon$ for any $\epsilon > 0$.*

We remark that this theorem holds no matter what decision procedure the algorithm uses to push/reject items in the push phase.

Proof. Consider an input sequence consisting of k identical *wide intervals*, and N^2 identical *narrow intervals*, for any integer $N > 1/\epsilon >> 1$. The wide intervals have slightly earlier finishing times than the narrow intervals, but overlap with the narrow intervals. The bandwidth and profit of the wide intervals are both $1/k$. The bandwidth of each narrow interval is $1/N^2$, and each narrow interval has profit of $1/N$. This initial set-up is depicted in Figure 5. By assumption, the algorithm considers all of the wide intervals first. If it ever rejects a wide interval, the adversary simply eliminates all of the narrow intervals. The total profit of the optimal solution is then 1, while the algorithm can now achieve at best a profit of $1 - 1/k$.

Assume, then, that the algorithm pushes all of the wide intervals onto the stack. It now must consider the narrow intervals. Suppose that at any point, it accepts a narrow interval. Then the adversary immediately eliminates the remaining unseen narrow intervals. During the pop phase, the one narrow interval on the stack is accepted, so one of the wide intervals will be deleted during this phase, since the total bandwidth of all of the wide intervals along with the one narrow interval is more than 1. Hence, the algorithm achieves a profit of only $1 - 1/k + 1/N < 1 - 1/k + \epsilon$, while the optimal profit is still 1. The only other option for the algorithm is to reject all of the narrow intervals , and achieve a profit of 1. In this case, however, taking all N^2 narrow intervals achieves an optimal profit of $N >> 1$. Thus, the stack algorithm cannot achieve a ratio of $1 - 1/k + \epsilon$ for any $\epsilon > 0$. $\square$

Note that for $k = 1$, the bandwidth problem, this result shows that, even if only one interval per job is allowed, no constant approximation factor is possible. In contrast, approximation ratios of 1/5 for

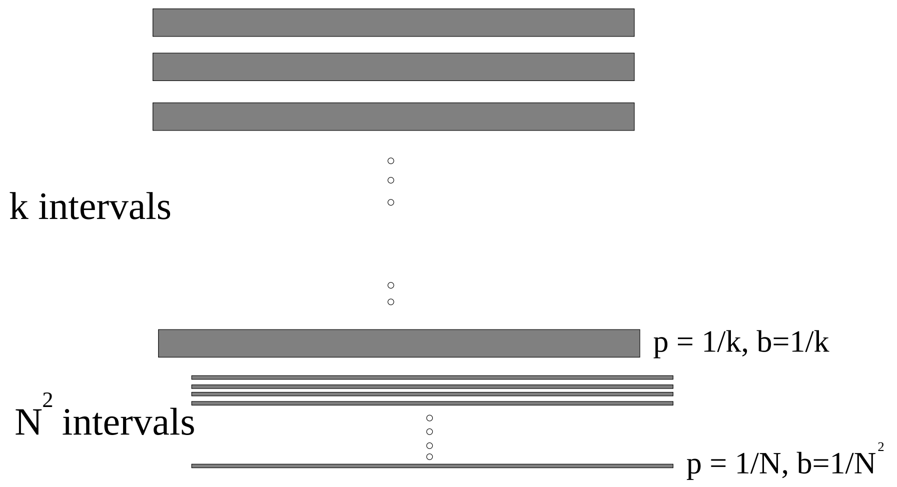

Figure 5: Initial set-up for theorem 17

the general bandwidth problem, and a ratio of 1/3 when there is only one interval per job, are given in Bar Noy et al [11]. These approximation ratios are obtained by running two local ratio (stack) algorithms and taking the best of these two results. This clearly does not fall within our model. As such, it does suggest the limitations of precise algorithmic models such as the one discussed in this paper. A more positive interpretation is that the limitations of any algorithmic model can suggest useful extensions. We remark that it is easy to modify Theorems 14 and 17 so as to prove $1-\Omega(1)$ inapproximation results for a model that allows taking the max profit of a constant number of stack algorithms, say $ALG_1, ALG_2 \ldots ALG_r$. Namely, the adversary initially presents r disjoint copies $\mathcal{I}_1, \mathcal{I}_2 \ldots \mathcal{I}_r$ of the input intervals used for the (single) stack algorithm inapproximation. (The constructions have to be modified so that the OPT in each copy is approximately the same.) The adversary then deletes intervals in each $\mathcal{I}_i$ to construct nemesis input sets as it would for each ALG_i. It follows that each algorithm will only receive a fraction of the optimal profit for one of the disjoint nemesis sets.

6 Discussion and Conclusion

We have presented a syntactic model that captures many primal dual/local ratio algorithms in the context of covering and packing problems. Our framework exposes limits of these paradigms and hence hopefully suggests new ways that modifications of these algorithmic techniques can be applied so as to obtain better approximation guarantees while still maintaining the syntactic and computational simplicity of the basic methods.

For example, our analysis of the interval scheduling problem does not preclude the possibility of nearly optimal and efficient (e.g. $O(n \log n)$ time) algorithms for a large but fixed number of processors. We have also seen the dependency of these methods on the input representation which corresponds to the constraints used in an LP relaxation of the problem. For the natural representations of set cover, Steiner tree and bandwidth allocation/interval allocation we can derive limitations on the approximation ratio of such algorithms. But our bounds suggest that further algorithmic improvements can be made even if we stay within the natural input representation.

Our stack framework also suggests some natural extensions to the known primal dual/local ratio paradigm; for example, allowing the stack algorithm to make irrevocable acceptances during the push phase that cannot be revoked during the pop phase. Another extension is suggested by the revocable acceptances priority framework of [46] that for example models the (job) interval scheduling algorithms in [38]. Namely, we may consider a packing stack model in which acceptances during the push phase can be revoked during the push phase (as well as in the pop phase). For the interval scheduling problem, we do not need to revoke acceptances during the push phase since every input interval has a unique set of conflicting intervals. It follows that for interval scheduling, any revocable acceptance priority algorithm can be simulated by a stack algorithm which simply delays any revocations until the pop phase. However, for other packing problems (for example, the knapsack problem) it seems that revoking items during the push phase may be necessary unless we allow a more complex pop phase. The stack model for covering or packing problems also suggests a "queue model" where items in the second phase are considered in FIFO rather than LIFO order. We note that both the stack model and the queue model are special (more tractable) cases of two pass priority algorithms. Finally, we argue that the stack framework clearly encourages us to think of alternative ways to order input items.

Acknowledgement
We greatly appreciate the careful and very constructive reviews by the referees. In particular, we thank the referees for their observations leading to Remark 4 as well as to many other improvements in the paper.

References

[1] Udo Adamy, Thomas Erlebach, Dieter Mitsche, Ingo Schurr, Bettina Speckmann, and Emo Welzl. Off-line admission control for advance reservations in star networks. In *WAOA*, pages 211–224, 2004.

[2] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SICOMP*, 24:440–465, 1995.

[3] Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. Opportunity cost algorithms for combinatorial auctions. *CoRR*, cs.CE/0010031, 2000.

[4] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. In *20th Annual IEEE Conference on Computational Complexity CCC05*, 2005. Journal version to appear in special issue of Computational Complexity, in memory of Michael Alekhnovich.

[5] D. Amzallag, R. Bar-Yehuda, D. Raz, and G. Scalosub. Cell selection in 4g cellular networks. In *Proceedings of the Annual IEEE 27th INFOCOM*, pages 700–708, 2008.

[6] S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. *Algorithmica*, 40(4):271–291, 2004.

[7] E. M. Arkin and E. L. Silverberg. Scheduling jobs with fixed start and end times. *Disc. Appl. Math*, 18:1–8, 1987.

[8] S. Arora, B. Bollobás, and L. Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of the 43rd Annual IEEE Conference on Foundations of Computer Science*, pages 313–322, 2002.

[9] S. Arora, B. Bollobás, L. Lovász, and I. Tourlakis. Proving integrality gaps without knowing the linear program. *Theory of Computing*, 2(2):19–51, 2006.

[10] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs. In *ISAAC Algorithms and Computation*, pages 142–151, 1995.

[11] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 48(5):1069–1090, 2001.

[12] Amotz Bar-Noy, Sudipto Guha, Yoav Katz, Joseph Naor, Baruch Schieber, and Hadas Shachnai. Throughput maximization of real-time scheduling with batching. *ACM Transactions on Algorithms*, 5(2), 2009.

[13] R. Bar-Yehuda, A. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approxmation algorithms: in memoriam Shimon Even 1935-2004. *Computing Surveys*, 36:422–463, 2004.

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

[15] R. Bar-Yehuda, M. M. Halldorsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 732–741, 2002.

[16] Reuven Bar-Yehuda, Michael Beder, Yuval Cohen, and Dror Rawitz. Resource allocation in bounded degree trees. *Algorithmica*, 54(1):89–106, 2009.

[17] Reuven Bar-Yehuda and Dror Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discrete Math.*, 19(3):762–797, 2005.

[18] Reuven Bar-Yehuda and Dror Rawitz. Using fractional primal-dual to schedule split intervals with demands. *Discrete Optimization*, 3(4):275–287, 2006.

[19] Ann Becker and Dan Geiger. Approximation algorithms for the loop cutset problem. In *Tenth Conference on Uncertainty in Aritificial Intelligence*, pages 60–68, 1994.

[20] P. Berman and B. DasGupta. A simple approximation algorithm for nonoverlapping local alignments (weighted independent sets of axis parallel rectangles). *Biocomputing*, 1:129–138, 2002.

[21] P. Berman and B. Das Gupta. Improvements in throughput maximization for real-time scheduling. In *STOC: ACM Symposium on Theory of Computing*, 2000.

[22] Dimitris Bertsimas and Chung-Piaw Teo. From valid inequalities to heuristics: a unified view of primal-dual approximation algorithms in covering problems. *Oper. Res.*, 46(4):503–514, 1998.

[23] A. Borodin, M. N. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.

[24] J. Buresh-Oppenheim, S. Davis, and R. Impagliazzo. A stronger model of dynamic programming algorithms. To appear in *Algorithmica*, 2010.

[25] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. An improved lp-based approximation for steiner tree. In *STOC*, pages 583–592, 2010.

[26] Gruia Calinescu, Amit Chakrabarti, Howard Karloff, and Yuval Rabani. Improved approximation algorithms for resource allocation. In *In 9th International Integer Programming and Combinatorial Optimization Conference IPCO*, pages 40 –414. Springer LNCS 2337, 2002.

[27] Deeparnab Chakrabarty, Nikhil R. Devanur, and Vijay V. Vazirani. New geometry-inspired relaxations and algorithms for the metric steiner tree problem. In *13th Integer programming and combinatorial optimization conference IPCO*, pages 344–358. Springer Lecture Notes in Computer Science 5035, 2008.

[28] Deeparnab Chakrabarty, Jochen Könemann, and David Pritchard. Hypergraphic lp relaxations for steiner trees. In *IPCO*, pages 383–396, 2010.

[29] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. Integrality gaps for Sherali-Adams relaxations. In *STOC'2009*. ACM Press, 2009.

[30] Miroslav Chlebík and Janka Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theor. Comput. Sci.*, 406(3):207–214, 2008.

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

[32] V. Chvátal. Hard knapsack problems. *Operations Research*, 28(6):1402–1441, 1980.

[33] Sashka Davis and Russell Impagliazzo. Models of greedy algorithms for graph problems. *Algorithmica*, 54(3):269–317, 2009.

[34] A. Dechter and R. Dechter. On the greedy solution of ordering problems. *ORSA Journal on Computing*, 1(3):181–189, 1989.

[35] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[36] J. Edmonds and J. Kwon. A faster approximate minimum steiner tree algorithm, May 2009.

[37] Thomas Erlebach and Frits C. R. Spieksma. Simple algorithms for a weighted interval selection problem. In *International Symposium on Algorithms and Computation*, pages 228–240, 2000.

[38] Thomas Erlebach and Frits C. R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *J. Algorithms*, 46(1), 2003.

[39] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[40] Uriel Feige and Robert Krauthgamer. The probable value of the Lovász-Schrijver relaxations for maximum independent set. *SICOMP: SIAM Journal on Computing*, 32(2):345–370, 2003.

[41] Wenceslas Fernández de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 53–61. Society for Industrial and Applied Mathematics, 2007.

[42] Konstantinos Georgiou, Avner Magen, Toniann Pitassi, and Iannis Tourlakis. Integrality gaps of 2-o(1) for vertex cover sdps in the lov[a-acute]sz–schrijver hierarchy. *SIAM J. Comput.*, 39(8):3553–3570, 2010.

[43] M. X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SICOMP*, 24:296–317, 1995.

[44] C. Gröpl, S. Hougardy, T. Nierhoff, and H.J. Prömel. Steiner trees in uniformly quasi-bipartite graph. *Information Processing Letters*, 83(4), 2002.

[45] Dorit Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. Course Technology, 1996.

[46] S.L. Horn. One-pass algorithms with revocable acceptances for job interval selection. *MSc Thesis, University of Toronto*, 2004.

[47] K. Jain and V. Vazirani. Approximation algorithms for the metric facility location problem and k-median problem using the primal-dual schema and lagrangian relaxation. *JACM*, 48:274–299, 2001.

[48] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. *SIAM Journal on Computing*, 28:164–191, 1998.

[49] P. Klein and R. Ravi. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.

[50] Jochen Könemann, David Pritchard, and Kunlun Tan. A partition-based relaxation for steiner trees. *CoRR*, abs/0712.3568, 2007.

[51] B. Korte and L. Lovász. Greedoids and linear objective functions. *SIAM Journal Algebraic and Discrete Methods*, 5:229–238, 1984.

[52] Kurt Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Inf. Process. Lett.*, 27(3):125–128, 1988.

[53] S. Rajagopalan and V.V. Vazirani. On the bidirected cut relaxation for the metric steiner tree problem. In *SODA*, pages 742–751, 1999.

[54] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 475–484, 1997.

[55] Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph steiner tree approximation. *SIAM J. Discrete Math.*, 19(1):122–134, 2005.

[56] H. Saran, V. Vazirani, and N. Young. A primal-dual approach to approximation algorithms for network steiner problems. In *Proceedings of the Indo-US workshop on Cooperative Research in Computer Science*, pages 166–168, 1992.

[57] Grant Schoenebeck, Luca Trevisan, and Madhur Tulsiani. Tight integrality gaps for Lovász-Schrijver LP relaxations of vertex cover and max cut. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 302–310. ACM, 2007.

[58] Mohit Singh and Kunal Talwar. Improving integrality gaps via chvátal-gomory rounding. In *APPROX-RANDOM*, pages 366–379, 2010.

[59] P. Slavík. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25:237–254, 1997.

[60] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

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

[62] Y. Ye and A. Borodin. Elimination graphs. In *36th International Colloquium on Automata, Languages and Programming ICALP 2009*, 2009. Journal version to appear in ACM Transactions on Algorithms TALG.

7 Appendix

In order to make the paper more self contained, we define the *hitting set problem* and include primal and dual programs for the hitting set problem as well as the related primal dual schemas as they appear in Williamson [61].

Hitting Set Problem: Given a ground set of elements E, non-negative costs c_e for all elements $e \in E$, and subsets $T_1, \ldots, T_p \subseteq E$, we want to find a minimum-cost subset $A \subseteq E$ so that A has a non empty intersection with each subset T_i.

$$\min \sum_{\mathrm{e}\in \mathrm{E}} \mathrm{c_e x_e}$$

subject to:

$$\sum_{e\in T_i} x_e \geq 1$$
$$x_e \geq 0$$

Figure 6: Primal LP for Hitting Set Problem

$$\max \sum_{\mathrm{i}=1}^{\mathrm{p}} \mathrm{y_i}$$

subject to:

$$\sum_{i:e\in T_i} y_i \leq c_e$$
$$y_i \geq 0$$

Figure 7: Dual LP for Hitting Set Problem

```
$y \rightarrow 0$
$A_1 \rightarrow \emptyset$
$\ell \rightarrow 1$($\ell$ is a counter)
While $A_\ell$ is not feasible
  Choose a set $\Upsilon_\ell$ of violated sets
  Increase $y_k$ uniformly for all $T_k \in \Upsilon_\ell$ until $\exists e_\ell \in T_k$
  such that $\sum i : e_\ell \in T_i y_i = c_{e_\ell}$
  $A_{\ell+1} \rightarrow A_\ell \cup \{e_\ell\}$
  $\ell \rightarrow \ell + 1$ $A' \rightarrow A_{\ell-1}$
For $j \rightarrow \ell - 1$ down to 1
  If $A' - \{e_j\}$ is still feasible
    $A' \rightarrow A' - \{e_j\}$
Return A'
```

Figure 8: The general primal-dual algorithm