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

Allan Borodin David Cashman†
Avner Magen

Department of Computer Science, University of Toronto

December 12, 2007

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. We provide tools to bound the performance 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 [7] 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 [1] and Goemans and Williamson [17].

The list of problems tackled successfully by primal-dual and local-ratio algorithm is long and impressive. For many of these problems the methods achieve the best known approximation (see [27, 6] and references therein). But just how far can these methods go? The goal of this paper is to shed some light on this question. Our first step in achieving this is to put the two algorithmic schemas under a common umbrella – a syntactic computational model capturing both, that is

*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

devoid of complexity considerations and that is related to the LP formulation via the input model that is used. This model is what we call the *stack model* and will be described in detail in Section 2. The stack model also captures the local-ratio schema as described in the survey of Bar-Yehuda et al [6]. This should not be taken as a surprise as Bar-Yehuda and Rawitz [9] have demonstrated that there is an equivalence between the primal dual and local-ratio schemas. We use the correspondence described there to argue that the *stack model* simulates local-ratio algorithms, and omit the explicit description of such a simulation.

For the clarity of our exposition we will use the set cover problem as a specific running example of a covering problem. We note that the set cover problem is a well understood problem relative to standard complexity assumptions (see Section 3) and hence we use this problem as an illustrative example rather than in the hope of deriving a better approximation algorithm. 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: our abstraction of this method corresponds to what Williamson [27] calls the "primal-dual algorithm with reverse delete step added" and its extension the "general-primal dual algorithm [1]". The more restricted version of the method which was the first to appear is called the "basic primal-dual algorithm", and can be simulated by the *priority model* of [11]. In the priority model, an algorithm considers each input item once (according to some ordering) and immediately makes an irrevocable decision (i.e. accept or reject) about each input item. For the simulation to be possible, the input items must correspond to the 0/1 variables of the problem; further they contain information about their cost and about the constraints they appear in. In the set cover example, the items are the sets and the natural representation of a set is its weight and the list of elements it contains, which are the cost and the constraints of the natural LP relaxation respectively. We elaborate on this somewhat subtle issue of input representation when we discuss the specific applications we deal with. Returning to priority algorithms, there is some natural restriction on what orderings are permissible and we shall argue that the primal dual schema always leads to a permissible ordering. Roughly speaking, the permissible orderings are those that are induced by any (not necessarily computable) mapping of all possible input items into $\mathbb{R}$. Also, the ordering in adaptive priority algorithms can be established afresh at every iteration of the algorithm.

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" [17, 20, 24]. Roughly speaking, this is a phase that guarantees that the output of the algorithm is a *minimal* feasible solution (with respect to set inclusion). This property is shown to be crucial in the analyses of problems such as minimum Steiner tree, the feedback vertex set problem, generalized Steiner forests, and others. As is shown in [27] the analysis may sometime depend not only on the minimality of the produced solution, but also on the exact process that ensures this minimality. The analogous leap in the world of abstract models leads us to the definition of the *stack model*. Here, instead of irrevocably rejecting/accepting an item, the algorithm decides whether to reject or to *push* an item onto a stack. Then the items in the stack are popped and deleted when this is possible.

In contrast to the extensive use of primal-dual and local-ratio for covering problems, the primary use of either schema for packing problems is the approximation of the *bandwidth allocation problem* in Bar-Noy et al [5]. As was shown in [5], the local ratio technique can *optimally* solve the one machine

[1] Williamson defines the primal dual method in terms of a general hitting set problem which in turn models a number of covering problems.

weighted interval selection problem whereas the negative result in [11] precludes the possibility of any constant approximation to the problem within the priority model. This provably shows that it is sometimes necessary to have a second "clean-up" stage to obtain good approximation bounds.

It is important to note that, while most 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 [19] use of primal dual method for the approximation of metric facility location and k-median problems [19]. Notice that these problems are not covering or packing problems, and therefore may inherently be in need of a more specialized algorithmic paradigm within the primal dual context [2]. A more basic example is the Steiner tree algorithm in [21] 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 ([8],[10]) which first solve an LP, and then use the LP solution as a guide to ordering items.

2 The Stack Model and its ancestor the Primal-Dual Schema

Consider an integer program for a covering problem, and assume that the variables to the problem are the natural 0/1 variables of the problem, and that the constraint matrix A and the cost vector c are nonnegative. The primal dual schema considers an LP relaxation of this IP and its dual maximization problem and proceeds as follows:

- (i) start with the feasible dual solution $\vec{y} = 0$
- (ii) increase components of $\vec{y}$ continuously until one (or more) dual constraint(s) becomes (simultaneously) tight
- (iii) when a dual constriant becomes tight, the corresponding primal variable enters the solution
- (iv) continue as long as the primal solution generated so far is not feasible
- (v) go over the primal variables in the solution in reverse order accepting variables needed for feasibility and deleting variables that are not needed for feasibility (given the variables already accepted and the variables remaining on the stack)

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

[2]The second phase of the Jain and Vazirani [19] primal dual algorithm uses a more involved process to delete certain facilities that were opened during the first phase.

We now get to the critical observation that later leads to the abstraction of the model and to our lower bounds. Notice that the above process induces, at each iteration, an ordering on the primal variables that have not yet entered the solution. If at any point an adversary announces that a primal variable that has not yet become tight *has never existed*, this has no effect on the runtime history of the algorithm has no way of such a change as it has no effect on the algorithm's history. An adversarial approach can be developed from this limitation of the algorithm and may lead to inapproximability results.

This sets the stage for the stack model. We consider a two-pass algorithm, in which the first pass, the push phase, resembles an adaptive priority algorithm, as defined in [11] and formalized in [13], and the second pass, the pop phase, is a simple greedy phase that is not under the control of the algorithm; here is a precise description. The algorithm first orders the input items by some valid ordering. This essentially means that the algorithm should commit to a total ordering of all possible items, and every instance then inherit the ordering induced o its particular items. In our example, one can imagine ordering by cardinality of sets, or more generally by any function f_1 on the characteristic 0/1 vectors and the weight of the sets. Next, the algorithm looks at the first item, and either pushes it onto the stack or *rejects* it. Using the information obtained from this first input item, th algorithm then chooses a new total ordering on the set of all possible remaining items (i.e. those that had lower priority than the first actual input item) which induces a new ordering on the actual set of remaining input items. For the set cover example, the algorithm can define f_2 after looking at the first set and knowing which elements are now covered. Again, the first (unseen) input item in this ordering is either pushed or rejected. This process continues for at most n iterations. At this point this algorithm enters the "pop phase" in which items are popped from the stack (last in, first out). A popped item will be *deleted* once it is popped if the items that were popped and accepted, together with the items still on the stack constitutes a feasible solution; otherwise it 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. Moreover, this solution is minimal with respect to set inclusion.

Having defined the stack model, we are ready to make the central claim of this section : Every primal dual algorithm under the scheme described above can be simulated by a stack algorithm. In order to claim that, we must be very clear about the asumption we make regarding the input representation: the information that an item holds should be (at the least) its cost, and the objects that correspond to the constraints in which it appears. In the standard LP for our set-cover example, this simlpy means that an item, namely a set, contains the information of its cost as well as the identity of the elements of that set. Indeed, regardless of the way y is increased, we get a primal variable that matches a dual constraint that was just made tight. The variable(s) made tight is the one (are those) minimizing the expression

$$f(i) = c_i - \sum_{j:a_{ij}>0} a_{ij} y_j.$$

But, since the information about item i includes its cost c_i and the positive constraint coefficients a_{ij}, and since the algorithm may keep track of y variables, we conclude that $f : [n] \mapsto \mathbb{R}$ is a valid ordering function as it is a function on the items. The first item is taken[3] and the stack algorithm pushes it and continues. The pop phase is clearly a reflection of step (v) in the primal dual schema.

[3]If some items become tight simultaneously this is simulated by taking those items one after the other

If the ordering used during the push phase is determined at the beginning and is not changed during the process, we call this a *fixed order* stack algorithm. Without this restriction we say the ordering is *adaptive*.

Following the discussion in Bar Noy et al [5], we now consider the formulation of the primal dual scheme for packing problems and the corresponding stack model. We consider the following primal dual framework similar to the one outlined for covering problems:

- (i) start with the (non-feasible) dual solution $\vec{y} = 0$
- (ii) increase components of $\vec{y}$ continuously until one (or more) selected unsatisifed dual constraint(s) become (simultaneously) satisfied; in doing so, other constraints may also get satisfied[4]
- (iii) when a dual constriant becomes satisfied, the corresponding primal variable enters the solution
- (iv) continue as long as there are unsatisfed dual constraints
- (v) go over the primal variables in the solution in reverse order accepting variables that do not violate the primal packing requirements.

Hence as in the stack model for covering problems, we have a push phase and a pop phase where the push phase proceeds like a fixed or adaptive priority algorithm in determining which items to place on the stack. The pop phase starts with the empty set as its existing solution, and then adds popped items to the existing solution unless the addition of the item renders the current solution infeasible.[5]

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 deleted 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.

[4] In the fixed order algorithm of [5], the primal variables/dual contraints 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 corresonding dual constriants to be satisfied.

[5] The reader may suspect that there should be a way to define a packing stack model as a "dual" to the covering model, exchanging the role of acceptances and rejections. This can indeed be done if we take a somewhat more general definition of the model where during the "push phase" we also allow the algorithm to permanently accept an item rather than just placing it on the stack as as "candidate for acceptance". This additional power does not seem to have been used in known algorithms and for the purposes of this paper we only allow permanent rejections in the push phase.

Remark 1 The reader may benefit from noticing as early as here, that there are three terms whose English meaning 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 an item that not witnessed by the algorithm actually does not appear in the input.

The reader may notice that in moving from the primal-dual methodology to the abstraction we describe, the linear structure of the (relaxed) problem disappears. One may also ask what role does the integrality gap play in this discussion. As was observed [27], the ratio guarantee of primal dual algorithms is never better than the integrality gap for *the specific linear program relaxation used by the algorithm*. However, *any* formulation of the problem as an LP with positive coefficients can lead to a primal dual algorithm, which in turn provides an ordering of the items that gives rise to a stack algorithm. In other words the abstraction helps us to understand the power of primal dual algorithms with respect to every LP relaxation with positive coefficients under the (nontrivial) condition that the variables are the "natural" ones, and that the input representation "knows" the cost of an item and the constraints involving it. Specifically, to show an inapproximability result for a suggested LP relaxation we are required to adjust the input-representation when we give the stack lower bound. In Vertex Cover, for example, we may look at the LP relaxation that uses the inequalities that require variables of the vertices in an odd cycle of size l to sum up to $(l+1)/2$ (on top of the regular inequalities). Notice that this type of tightening is a result of a lift and project construction and was discussed in the context of hardness result in [4]). If we manage to get a lower bound for an input representation in which a vertex knows all the edges *and* all the odd cycles containing it, this would imply a lower bound for a primal dual algorithm applied to this relaxation!

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 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 [22] 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 [16] 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 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 [2].

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 [26] 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 $\mathbb{GF}_2$. 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 a pair of complementary sets.

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 2 *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 iff it is a solution of the system $\langle v_1, x\rangle = 1 - b_1, \ldots \langle v_k, x\rangle = 1 - b_k$. Since the v_i are linearly independent, the co-dimension of the solution space of uncovered elements is k and so it contains $|F_2|^{l-k}$ elements and the first part of the lemma follows. The second part is obvious. □

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 $S^{b_i}_{v_i}$. Then, the adversary eliminates all the sets S^b_v 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^0_v and S^1_v cannot both be on the stack.

In the other case in which the algorithm rejects the set $S^{b_i}_{v_i}$, 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 $S^{1-b_i}_{v_i}$. 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 2 does not form a cover. But of course one exists as the input contains the two complementary sets $\{S^{b_0}_{v_i}, S^{b_1}_{v_i}\}$. 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 $S^0_{v^*}$ and $S^1_{v^*}$, where

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

(these sets were not eliminated before as $v^* \notin \text{span}\{v_1, \ldots, v_{\log n - 1}\}$). 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 $S^0_{v^*}$ and $S^1_{v^*}$ 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 the corollary 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 3 *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 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 [9]. Pick an edge with minimum $w_e/|e \cap T|$, recursively solve the problem in which that edge is contracted to a vertex, noticing that such a solution with the additional edge e is a solution to the original problem. The reverse delete step ensures that if an edge is not needed it will not be taken. In fact, here this step is easily seen to be essential, and 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 outside of the scope of our stack model? Unlike the set cover problem, it is still plausible (relative to known comlexity based hardness results) that there is a conceptually simple and efficient approximation algorithm that improves upon currently known polynomial time approximation algorithms. In [25], it is shown that unless Co-$RP = NP$, no approximation ratio better than about 1.007 is possible for the Steiner Tree problem. The best known algorithm for the problem, due to Robins and Zelikovsky [23] achieves approximation ratio of 1.55 and is not a local-ratio/primal-dual algorithm. For the class of quasi-bipartite graphs (i.e. graphs not containing edges between Steiner nodes), their algorithm achieves a ratio of about 1.28, slightly better than our 4/3 lower bound (derived in Theorem 6) for any stack algorithm in the standard input model. We note that our proofs apply to quasi-bipartite graphs. In some contexts this shows that the algorithm in [23] is superior to any primal-dual with reverse delete algorithm. However, unlike the set cover problem, it is still plausible (relative to known comlexity based hardness results) that there is a conceptually simple (e.g. primal dual) and efficient approximation algorithm for the general Steiner tre problem that improves upon currently known polynomial time approximation algorithms.

We will present lower bounds for both input models motivated by the priority lower bound of Davis and Impagliazzo [13]. For both input models we will consider bipartite graphs $G =< T \cup S, T \times S >$, where T is the set of terminal nodes and S is the set of Steiner nodes. (Obviously these nemesis graphs are quasi-bipartite.) We make the following simple observation for such bipartite graphs:

Observation 4 *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 5 *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 6 *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 Steiner trees must use both Steiner nodes and have cost 4.

Using Observation 4, 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 5 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 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 7 *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 8 *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 5 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 rooted at a Steiner nodes $s' \neq s$ and has cost 4 while the algorithm's solution must either be a tree 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 9 *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.*

Let $< e_1, e_2, e_3 >$ be the first three edges considered by the algorithm (in that order) during the push phase. In view of Lemmas 7 and 8, we fix our attention to the case where the first three edges e_1, e_2, e_3 are pushed and touch all Steiner nodes. Note also that the advesray has not yet intervened. Assume e_i ias adjacent to node s_i. We consider two cases:

Case 1: The edge costs (as a set) 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 vertex s_1) and

let the cost of e_2 be no greater than the cost of e_1 (which is possible since we are not in the $\{1,2,2\}$ situation. Using Lemma 7, we can assume that the algorithm has pushed e_4. Let $d \in \{1,2\}$ be the cost of e_1. Then the adversary 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 5, 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 7. 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 5, 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 [5] provide the primary use to date of a local-ratio/primal dual approximation algorithm for a packing problem. Several problems are discussed in [5], 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.

We first consider the (polynomial time solvable) special case of weighted interval scheduling on m identical machines ($m-WISP$) where (in terms of the bandwidth problem) every interval has bandwidth equal to $\frac{1}{m}$. That is, the optimization prolem 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[6] 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 [5]. We also note that an optimal algorithm for $m-WISP$ can be obtained by an $O(n^m)$ time

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

dynamic programming algorithm or a time $O(n^2 \log n)$) min cost max flow based algorithm [3].

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 [15]. We emphasize that our bound does not require that this is the ordering used by the algorithm.

Theorem 10 *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 10 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 [5], 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 [12], 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 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 start by describing the *forbidden configurations* of the input that we will use.

The first forbidden configuration (FC1) consists of intervals I, J, K that appear in that order with $I \cap J \cap K \neq \emptyset$; further they have profits x, y, z respectively that satisfy $y < x, y < z$. We claim that for such a configuration (namely, an input consisting of at most these three intervals, and an ordering as described above) the best approximation ratio achievable is

$$\max\{x/(x+y), (x+y)/(x+z), (y+z)/(x+z)\}. \tag{1}$$

To see the claim, consider the action of the algorithm on an input that contains (at most) I,J and K. If the algorithm decides to reject an interval, the adversary eliminates all remaining intervals. This leads to approximation ratios $0, x/(x+y)$ or $(x+y)/(x+z)$ 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 $y + z$ while $x + z$ is possible, and the bound (1) follows.

The second forbidden configuration (FC2) we consider is the following: there are four intervals I_1, I_2, J, K with profits x, x, y, z respectively. I_1 and I_2 are disjoint, and $I_i \cap J \cap K \neq \emptyset$ for $i = 1, 2$.

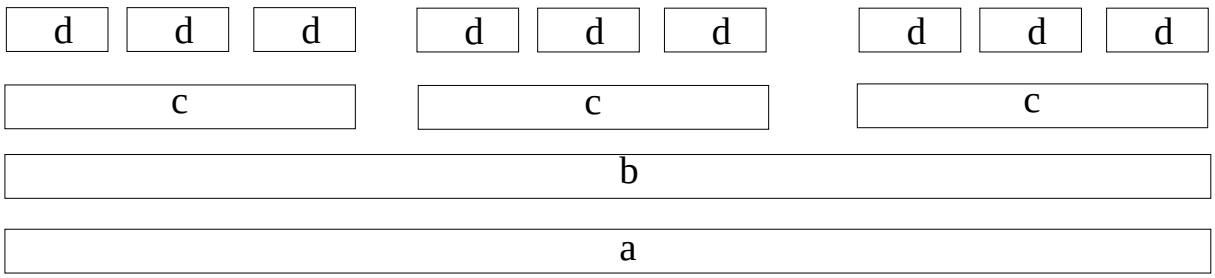

Figure 1: Initial input for interval scheduling on two machines used for Theorem 10 (with the corresponding profits)

Moreover, $x < y < 2x, y < z$ and the ordering is either I_1, I_2, J, K or K, J, I_1, I_2. We give an inapproximation bound against this configuration. 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 $2x + z$ the algorithm achieves $\max\{y + z, 2x + y\}$. Now suppose some intervals get rejected, and assume first that the ordering is I_1, I_2, J, K. If the adversary react to any rejection by eliminating all remaining intervals, this gives upper bound on the approximation of $0, x/2x, 2x/(2x + y), 2x + y/2x + z$ 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 $0, z/(z + y), (y + x)/(2x + z)$ 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 $2x+z$ the algorithm either obtains profit $y+x$ by pushing I_2 or (the better) profit $y+z$ by rejecting I_2. To sum up, the configuration FC2 implies an upper bound on the approximation of

$$\max\{1/2, 2x/(2x + y), (2x + y)/(2x + z), z/(y + z), (y + z)/(2x + z)\}. \tag{2}$$

Noting that $1/2 < z/(y + z) < (y + z)/(2y + z) < (y + z)/(2x + z)$, this simplifies to

$$\max\{2x/(2x + y), (2x + y)/(2x + z), (y + z)/(2x + z)\}. \tag{3}$$

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 1. We will fix the variables a, b, c, d that determine the profit 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 x, y, z in equations 1 and 3, 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:

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

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 commnet 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 or 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 $\frac{\sqrt{30}}{6} \approx 0.913....$

□

Returning to the more general bandwidth allocation problem, as previously mentioned the local ratio method in [5] 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 11 *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.

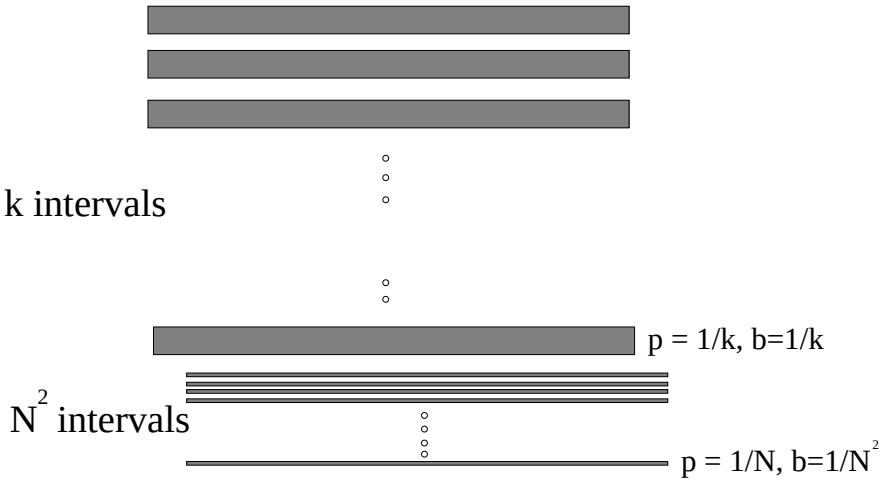

Figure 2: Initial set-up for theorem 11

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 2. 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 $\epsilon > 0$. □

Note that for $k = 1$, the general bandwidth problem, this result shows that, even if only one interval per job is allowed, no constant approximation factor is possible. Interestingly, in [5] an approximation ratio of 1/5 is shown for the general bandwidth problem, and a ratio of 1/3 is given when there is only one interval per job. However, these algorithms rely on running a local ratio algorithm twice and hence do not fall into our model. However, it does suggest the limitations of formal algorithmic models such as the one discussed in this paper, since a simple extension of an algorithmic model can produce significantly improved results. However, a more positive interpretation is that the limitations of any algorithmic model can suggest useful (but hopefully still analyzable) extensions.

6 Discussion and Conclusion

We have presented a syntactic model that captures the standard use of 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. For the more general bandwidth allocation problem, we would like to be able to derive a stack algorithm that can yield a ratio better than 1/2 for small bandwidths. 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 improvements can be made even assuming 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 [18] that for example models the (job) interval scheduling algorithms in [14]. 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. However, for other packing problems it seems that revoking items during the push phase may be necessary unless we allow a more complex push phase.) The stack model for covering or packing problems also suggests a "queue model" where items in the second phase are cnsidered 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 algorithms. Finally, we argue that the stack framework clearly encourages us to think of alternative ways to order input items.

References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] R. Bar-Yehuda and D. Rawitz. Using fractional primal-dual to schedule split intervals with demands. In *Proceedings of the 13th Annual European Symposium on Algorithms ESA, volume 3669 of Lecture Notes in Computer Science, Springer-Verlag*, pages 714–725, 2005.

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

[12] M. Alekhnovich A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. In *Proceedings of Computational Complexity Conference (CCC)*, pages 308–322, 2005.

[13] S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[14] T. Erlebach and F.C.R. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Technical Report 152, Computer Engineering and Networks Laboratory, ETH*, October 2002.

[15] 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.

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

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

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

[19] 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.

[20] 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.

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

[22] 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.

[23] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *SODA*, pages 770–779, 2001.

[24] 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.

[25] Martin Thimm. On the approximability of the steiner tree problem. In *Lecture Notes in Computer Science*, page 678. Springer-Verlag Heidelberg, 2001.

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

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