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

Allan Borodin, David Cashman, and Avner Magen

Department of Computer Science ⋆
University of Toronto

Abstract. We define an algorithmic paradigm, the stack model, that captures most primal-dual and local-ratio algorithms for approximating covering and packing problems. The stack model is defined syntactically and without any complexity limitations. hence our approximation bounds are independent of the P vs NP issue. 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.

⋆ {bor|cashman|avner}@cs.toronto.edu

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 [15].

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 [24, 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 larger umbrella – a syntactic computational model capturing both, devoid of complexity considerations and that is related to the LP formulation via the input model environment it works over. This model is what we call the *stack model* and will be described in detail in Section 2. We then show hardness of approximation results within this model for a few prominent problems in which the primal-dual schema was applied before, sometimes achieving the best bound.

For the clarity of our exposition we will use the set-cover problem as a specific running example. We shall focus our attention on what is considered to be the common method of the primal-dual schema for covering problems. Let us be more specific: our abstraction of this method corresponds to what Williamson [24] calls the "primal dual algorithm with reverse delete step". The more restricted version of the method which was the first to appear is called "basic primal-dual algorithm", 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 are in correspondence with the 0/1 variables constituting the problem, and the input items 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, namely the cost and the constraints of the natural LP relaxation. We elaborate on this somewhat subtle issue 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 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, namely the introduction of the "reverse delete step" [15, 17, 21]. Roughly speaking, this is a phase that guarantees that the output of the algorithm is a *minimal* feasible solution (w.r.t 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 [24] 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 elements in the stack are popped and removed when this is possible.

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. In fact, in this extended abstract we use their observation to argue that the *stack model* simulates regular local-ratio algorithms, without a concrete description of such a simulation.

In contrast to the extensive use of primal-dual and local-ratio for covering problems, the only (albeit important) use of the primal dual schema for packing problems was 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 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 use of primal dual method for the approximation of metric facility location and k-median problems [16]. Notice that these problems are not covering or packing problems, and therefore inherently in need of a more specialized algorithmic paradigm. A more basic example is the steiner tree algorithm for the special class of quasi-bipartite graphs [18]. 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 which first solve the LP, and use the solution as a guide to ordering items [8] and [10].

2 The Stack Model and its ancestor the Primal-Dual

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. In a nutshell, what a primal dual schema does is to consider an LP relaxation of this IP and the dual maximization problem and (i) starts with the feasible dual solution $y = 0$ (ii) increases y continuously until a dual constraint(s) becomes tight (iii) in this case the corresponding primal variable(s) enter the solution (iv) continues as long as the primal solution generated so far is not feasible (v) goes over the primal variables in the solution in a reverse order and removes them whenever feasibility is maintained. Notice that we haven't specified exactly how the dual vector y is increased, and we choose to stay at this level of generality. Returning to our set-cover example, the dual variables in the natural LP relaxation are elements of the ground set. We increase all the dual variables that do not occur in a tight equation (uncovered elements) uniformly, until a new set (dual constraint) becomes tight, in which case this set joins the solution. Step (v), the reverse-delete, removes a set if it is not needed, while following the reverse order.

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 has not entered the solution yet. If at any point an adversary announces that a primal variable that has not yet become tight *has never existed*, the algorithm has no way to recognize 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 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 items by some valid ordering. This is just a total ordering of all possible input 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, it looks at the first item, and either pushes it onto the stack or rejects it. It then may decide to change the ordering using the information of the first item (defining f_2 after looking at the first set). Again, the first (unseen) item in this ordering is either pushed or rejected. This process continues for at most n iterations, [1] at which point the algorithm enters the "pop phase" in which items are popped from the stack (last in, first out). The solution to be produced is a subset of the items on the stack and is defined as follows. Each popped item will be rejected if the items that were popped and accepted, together with the items still on the stack constitutes a feasible

[1] As we shall see later, the stack algorithm can end the push phase once a feasible solution exists on the stack, as the items past this point are bound to get deleted

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 the end, and further, that solution is minimal w.r.t 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. It is here that we insist that the input representation of an item contains its cost and the objects that correspond to the constraints that include it. 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 c_i and the positive a_{ij}, and since the algorithm may keep track of y 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 [2] and the stack algorithm pushes it and continues. The pop phase is clearly a reflection of step (v) in the primal dual schema.

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*.

We now consider the modifications needed in order to define a stack model for packing problems. Here too, 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. [3]

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 delete input items. 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

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

[3] 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.

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 deleted 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.

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 [24], 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 elements 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 constrains 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 method 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 PD algorithm applied to this relaxation!

3 The minimum set cover problem

We consider the classical 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 is associated with 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 special case of a stack algorithm.

Raz and Safra [19] 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 was able to show an almost tight bound $(1-\epsilon)H(n)$. But these hardness results do not apply to our model as the stack algorithm is not restricted to run in polynomial time. 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 a reformulation of a similar result from [2].

Let the ground set be the discrete cube $\{0,1\}^l$ where $l = \log n$, and let the initial input be the $2l$ sets corresponding to the facets of the cube, in other words, all sets $\{x \in \{0,1\}^l : x_i = b\}$ where $i = 1, \ldots, l$ and $b \in \{0,1\}$. Notice that this input consists of n pairs of complementary sets. Assume wlog 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 removes the set $\{x : x_1 = 1\}$. At this point we remain with the exact same problem, but with l decreased by 1. Since for $l = 1$ there are 2 sets that need to be accepted, we get that there are $l + 1$ sets that the priority takes instead of just two. This gives the desired bound.

The above demonstrates 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 [23] 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 $v \in \{0,1\}^{\log n} \setminus \{0\}$, 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 1. *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 by $\{S_{v_1}^{b_1}, \ldots, S_{v_k}^{b_k}\}$ are selected is exactly $n/2^k$. In particular, any family of sets whose corresponding vectors are linearly independent does not cover U completely.*

Using the set-system above, we will show that no stack algorithm can achieve a set cover smaller than $\log(n+1)$, 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 remove 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 removes all the sets S_v^b with $v \in \mathrm{span}\{v_1, \ldots, v_i\}$. 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 deletes all unseen inputs except for $S_{v_i}^{1-b_i}$. Note that $v_i \notin \mathrm{span}\{v_1, \ldots, v_{i-1}\}$ 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 1 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 delete all sets *except* for $S_{v^*}^0$ and $S_{v^*}^1$, where

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

(these sets were not removed before as $v^* \notin \mathrm{span}\{v_1, \ldots, v_{\log n-1}\}$). 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 $S_{v^*}^0$ 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 rejected 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 $v_1, \ldots v_l, v*$ 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 removed in the pop phase. We have established

Theorem 1. *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 a 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. The input representation we consider is with edges as items. The information in an item is its weight and the identity of the vertices of the edge. Also, the names of the vertices is known and whether they are terminals or not. Notice that this allows us to know all the separating sets an edge crosses. We later mention that a lower bound under a much richer input representation is possible, and will discuss the resulting implication. 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 w.r.t set inclusion. In this section we show that stack algorithms cannot approximate the problem with better than a 4/3 factor.

What is known about the Steiner tree problem outside of the scope of our stack model? In [22], 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 to the problem, due to Robins and Zelikovsky [20] 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 lower bound of 4/3 for any stack algorithm. We note that our proof applies to quasi-bipartite graphs. In some sense this shows that the algorithm in [20] is superior to any primal-dual approach. We now turn to our result.

Theorem 2. *No stack algorithm can achieve a worst-case approximation ratio better than* 4/3 *for the (unweighted) Steiner tree problem.*

Proof. Our proof is motivated by the priority lower bound of Davis and Impagliazzo [13]. Initially we consider the complete bipartite graph $G = < T \cup S, R \times S >$, where $T = \{t_1, t_2, t_3\}$ are the terminals and $S = \{s_1, s_2\}$ are the steiner nodes. The algorithm is given the set of terminals and steiner nodes in advance, but is not provided with a list of edges. After each stage of the algorithm, the adversary has the option of deleting any edges that have not yet been seen.

The main ingredient of the proof is the following lemma.

Lemma 2. *In the push phase, consider the first time t that the algorithm has seen two edges connected to some s_j assuming it has not rejected any edges before time t. Call this event D. Then the solution produced by the algorithm must contain these two edges.*

Observe that the stars rooted at s_1 and s_2 both have cost 3, while the other steiner trees must use both steiner nodes, and will have cost 4. Note that if an edge adjacent to either steiner node is missing (i.e. deleted by the adversary) then the star rooted at the other steiner node will be the only Steiner tree with cost 3. Here then is the lower bound strategy. The adversary lets the algorithm run, and waits until either the algorithm rejects an edge, or event D happens.

Suppose that the algorithm rejects an edge before event D happens, say the one connected to s_1. Then the adversary simply deletes an unseen edge adjacent to s_2. Such an edge must exist, since event D has not occurred yet. This makes the star rooted at s_1 the unique optimal solution, but the algorithm has just rejected an edge in that star. Hence, in this case the algorithm can achieve a cost of at best 4, for a ratio of 4/3.

Now, suppose that event D happens first. The algorithm pushes the two edges connected to, say, s_1. At this point the adversary deletes the third edge adjacent to s_1, leaving the star rooted at s_2 as the only optimal solution. But by Lemma 2, the algorithm cannot return this unique optimal solution , since it is bound to take both edges connected to s_1 in the solution. This again leads to an approximation lower bound of 4/3.

By a somewhat more involved example and analysis we can prove a 7/6-approximation lower bound when we allow the algorithm to not only know in advance the nodes of the graph, but also the edges of the graph. The only thing not known in advance is the edge costs. This 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.

5 Packing problems: scheduling

As was mentioned in the introduction, Bar Noy et al [5] provide the only use to date of local-ratio/primal dual for packing problems. Several problems are discussed there, the most general one being the NP-hard bandwidth allocation problem. Here we concentrate on one (polynomial time solvable) special case, namely weighted interval scheduling on 2 identical machines ($WISP_2$), and show that 0.913 is an upper bound [4] to the approximation of this problem by a fixed order (packing) stack algorithm. This should be contrasted with the optimal local ratio algorithm for one machine interval scheduling (WISP_1) and the 2/3-approximation supplied in [5] for $WISP_2$. We also note that an optimal algorithm for $WISP_m$ can be obtained by a time $O(n^m)$ dynamic programming algorithm or a time $O(n^2(n-m)$ min cost max flow based algorithm [3].

For the interval scheduling problem (and 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 weights (profits). This representation is thus good "against" the natural LP formulation in which

[4] As this is a maximization problem we will present approximation ratios to be less or equal to 1 and hence an "upper bound"on the ratio is a negative result.

the constraints are the bounds on the number of intervals that can be scheduled at any 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 [14]. Obviously our bound does not require that this is the ordering used by the algorithm.

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 the Ramsey phenomena in graphs; i.e. in every colouring we can say something about one of the colour classes. Restricted to such a (now ordered) subset, we are able to bound the quality of the algorithm (see [12] Theorem 4.1).

We start by describing *forbidden configurations* of the input with respect to an ordering. The first forbidden configuration are intervals I, J, K that appear in that order and with profits x, y, z respectively, so that $I \cap J \cap K \neq \emptyset$, and so that $y < x, z$. We claim that if there is such a configuration, 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 input that contains (at most) only I,J and K. If the algorithm decides to reject an interval, that interval becomes the last interval of the input. 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 rejected as it will be popped when J and K are already in the solution. Therefore the algorithm achieves $y+z$ while $x+z$ is possible, and the bound 1 follows.

Another forbidden configuration we consider is the following. There are four intervals I_1, I_2, J, K of profits x, x, y, z respectively. I_1 and I_2 are disjoint, and $I_i \cap J \cap K \neq \emptyset$ for $i = 1, 2$. Also assume that $y < 2x, z$ and that the order is I_1, I_2, J, K or K, J, I_1, I_2. By a slightly more careful yet similar analysis, we show that this configuration implies a lower bound of

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

We now consider a laminar set of intervals, with containment relation depicted in figure 1 by a rooted tree: an interval of a node is contained in the intervals of its ancestors, and otherwise two intervals are disjoint. Further, the profits of the intervals correspond to the level in the tree of the vertex representing the interval, and is (from top to bottom) a, b, c, d. We will later fix these vales and we only require here that $a > b > c > d$ and $2d > b$. We again can define a set of forbidden configurations for such a set of intervals. We will show that either a d or a c interval appears between two other intervals of greater profit (configuration I) or a b-interval appears between a-interval and two c-interval or a c-interval appears between a b-interval and two d-intervals (configuration II). By appropriately substituting the a, b, c, d values for x, y, z in equations 1 and 2, and deleting dominated terms (i.e. those that are guaranteed by the inequality conditions on a, b, c, d not to be the maximum), we obtain the following bound on the approximation factor achieved by any algorithm that chooses an ordering containing one of these forbidden configurations:

$$\begin{aligned} \max\ \{&a/(a+d), (a+c)/(a+b), (a+d)/(a+c),\\ &(c+d)/(b+c), 2c/(2c+b), (2c+b)/(2c+a),\\ &(a+b)/(2c+a), 2d/(2d+c), (2d+c)/(2d+b)\} \end{aligned} \tag{3}$$

We now sketch the proof of the inevitability of the forbidden configurations. Assume configuration I is avoided. In this case the a and b intervals must appear consecutively in the order. At least two of the c-intervals will then appear after them or before them; we will delete the other c intervals and its d-interval children. Another fact implied by the absence of configuration I is that each d-interval must appear before or after all of its parent intervals. Consider the d intervals of the c-interval that comes farthest away from a and b in the ordering. (That is, if the two remaining c-intervals are before a and b in the ordering, then take the c-interval that is ordered earlier, and if they are after, then take the c-interval that is ordered later.) Since they must appear before or after their three parent intervals, two of these d-intervals must appear at the head or tail of the ordering. We delete the rest of the d-intervals. At this point we know that there is an ordering of a, b, c, c, d, d intervals in which the pairs $a - b$, $c - c$ and $d - d$ are all consecutive. It is easy to see that there are four possible arrangement that avoid configuration I (left is early in the order) : $d - d - c - c - a - b$ or $d - d - b - a - c - c$ or $c - c - a - b - d - d$ or $b - a - c - c - d - d$. It now remains to notice that all these arrangements contain configuration II : $d - d - c - a$, $d - d - b - a$,$a - b - d - d$ and $a - c - d - d$ respectively. 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 of the approximation over all possible configurations. We optimize to get $(a, b, c, d) = (10, 8, 3\sqrt{30} - 10, 5)$ which leads to a lower bound of 0.913.

Theorem 3. *For interval scheduling on two machines, no fixed ordering stack algorithm can achieve a constant approximation factor better than* .913.

We remark that the above can be extended to any number k of machines, and leads to a $1 - O(1/k)$ inapproximability result. This is not satisfying as the upper bound for this case by local ratio [5] is $\frac{1}{2-1/k}$ for $k \geq 2$ which limits to $1/2$ as k gets large.

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 close to 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 smaller 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. The framework also encourages us to think of other reasonable 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. the power of priority algorithms for facility location and set cover, 2002.
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, 2004.
11. A. Borodin, M. N. Nielsen, and C. Rackoff. (Incremental) priority algorithms. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.
12. M. Alekhnovich A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, and T. Pitassi. Toward a model for backtracking and dynamic programming. *Unpublished manuscript*, 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. M. X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SICOMP*, 24:296–317, 1995.
16. 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.
17. 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.
18. Rajagopalan and Vazirani. On the bidirected cut relaxation for the metric steiner tree problem. In *SODA*, pages 742–751, 1999.
19. 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.
20. G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *SODA*, pages 770–779, 2001.
21. 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.
22. Martin Thimm. On the approximability of the steiner tree problem. In *Lecture Notes in Computer Science*, page 678. Springer-Verlag Heidelberg, 2001.
23. V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.
24. D. P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming, Series B*, 91(3):447–478, 2002.

7 Appendix

7.1 Proof of Lemma 1

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.

7.2 Proof of Lemma 2

Proof. The lemma is not vacuous : for the graph to have a steiner tree, at least one of the steiner node must have degree at least 2. These edges must be pushed since at time t there is no way to preclude the possibility that the graph is a path of length 4 (this is still consistent with the input so far) and therefore all edges must be taken, and a reject is impossible.

To show that these two edges must be used in the solution produced by the algorithm, we should show that there is no edge e in this pair that will be not rejected in the pop phase. Assume otherwise, then by the definition of the pop phase, there must be a cycle C containing e, so that e is pushed last among all edges in C. But since the graph is bipartite, C must contain two edges out of the other steiner node. This leads to a contradiction, since the later of these edges to be considered must come after e in the order.

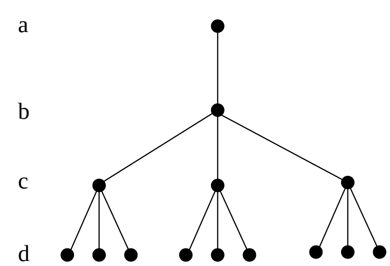

Fig. 1. Initial configuration of input for interval scheduling on two machines