

An Incremental Model for Combinatorial Maximization Problems

Jeff Hartline and Alexa Sharp

Department of Computer Science, Cornell University, Ithaca, NY 14853
{jhartlin, asharp}@cs.cornell.edu

Abstract. Many combinatorial optimization problems aim to select a subset of elements of maximum value subject to certain constraints. We consider an incremental version of such problems, in which some of the constraints rise over time. A solution is a sequence of feasible solutions, one for each time step, such that later solutions build on earlier solutions incrementally. We introduce a general model for such problems, and define incremental versions of maximum flow, bipartite matching, and knapsack. We find that imposing an incremental structure on a problem can drastically change its complexity. With this in mind, we give general yet simple techniques to adapt algorithms for optimization problems to their respective incremental versions, and discuss tightness of these adaptations with respect to the three aforementioned problems.

Keywords: analysis of algorithms, approximation techniques, combinatorial problems, network analysis, online problems

1 Introduction

There has been recent interest in incremental versions of classic problems such as facility location [1], k -median [2], maximum flow [3], and k -centre [4]. These problems model situations in which there is a natural hierarchy of levels with different characteristics, such as local vs. wide-area networks or multilevel memory caches. Incremental variations of NP-hard problems contain their non-incremental versions as special cases and therefore remain NP-hard. It is interesting to ask whether incremental versions of polytime problems remain polytime, or whether the incremental structure alters the problem enough to increase its complexity.

Traditional algorithms require all input at the outset and then determine a single solution. In practice, however, many problems require solutions to be built up over time due to limited resources and rising constraints. In this scenario, one or more of the inputs to the problem increases at discrete time intervals. One wants the best solution at each step, subject to the constraint that elements in the current solution cannot be removed. We would like a good solution at all stages; however, a commitment at some stage may limit the options at later stages. It is this tension between local and global optimality that makes these incremental problems challenging.

For example, consider the routing of a message from a source to a destination in an untrusted network. To increase security and preserve power at the internal router nodes, it is desirable to divide the message among node-disjoint paths, as proposed

by Lou and Fang [5]. Adversaries within the network must intercept all components of a well-divided message before they are able to decipher it. Also, transmitting a message along many node-disjoint paths prevents individual routers from depleting their battery faster than their neighbors. Thus more node-disjoint paths available in the network results in an increase in data confidentiality and energy efficiency.

Now suppose the underlying network changes over time: transmission distances increase and links appear between existing routers. We still want as many node-disjoint paths as possible, but we do not want to reprogram routers. More specifically, we would prefer not to change any of the existing node-disjoint paths.

The above problem is an application of the incremental maximum flow problem. The *incremental max flow* problem is defined on a directed network with source s , sink t , and a non-decreasing sequence of capacity functions, one for each time step. The problem is to find a sequence of s - t flows such that each flow satisfies its corresponding capacity constraints but does not remove flow from any prior solution. For each time step ℓ , we compare the value of the ℓ^{th} flow to the value of the optimal flow satisfying the current capacity constraints. We want this ratio to be close to 1, but because we are constrained to keep all flow from previous time steps, this may not be possible. Therefore one goal is to find a sequence of flows that maximizes the minimum of this ratio over all values of ℓ . An algorithm for this problem is said to be *r-competitive* if the minimum of the ratio over all ℓ is no less than r . This value of r is called the *competitive ratio* of the algorithm. Alternatively, if we are more interested in overall performance rather than the performance at each time step, our goal would be to maximize the sum of the solutions over all time steps. In our routing example, this objective would correspond to maximizing the throughput over all time.

Following this framework, we can define incremental versions of other combinatorial maximization problems. The *incremental bipartite matching* problem is defined on a bipartite graph where edges appear over time. A solution for this problem is a sequence of matchings, one for each time step, such that each matching contains all previous matchings. This model can be applied to job scheduling, as it can be costly or disruptive to reassign jobs. In *incremental knapsack* we are given a set of items with sizes and a sequence of increasing knapsack capacities. We want a sequence of knapsack solutions such that each solution contains all the items in the previous solution. Memory allocation is one application of incremental knapsack.

Our Results. We introduce a general incremental model and analyze the complexity of the three incremental problems defined above with respect to two different objective functions: maximum ratio and maximum sum. We find that the incremental version of bipartite matching remains polytime in many cases and becomes NP-hard in others, whereas the corresponding model of the max flow problem becomes NP-complete even for very basic models. Our central contribution is a general technique to translate exact or approximate algorithms for non-incremental optimization problems into approximation algorithms for the corresponding incremental versions. We find that these techniques yield tight algorithms in the case of max flow, but can be improved for bipartite matching and knapsack. The best known approximation bounds are given in Table 1.

The incremental model is laid out in Section 2. We present complexity results for the max sum objective in Section 3 and analogous results for the max ratio objective in Section 4.

Related Work and Extensions. Several incremental problems have been studied recently; Mettu and Plaxton [2] study incremental versions of the uncapacitated k -median problem and give a 29.86-competitive algorithm. Plaxton [1] introduces the incremental facility location problem and gives a $(1 + \epsilon)\alpha$ -competitive algorithm, given an α -approximation to the uncapacitated facility location problem. This results in a 12.16-competitive algorithm. Gonzales [6] gives a 2-approximation algorithm for the k -centre problem, which is also a 2-competitive algorithm for the incremental version. Lin et al. [4] present a general framework for cardinality constrained minimization problems, resulting in approximation algorithms for k -vertex cover and k -set cover, an improved approximation algorithm for incremental k -median, and alternative approximation algorithms for incremental k -MST and incremental facility location.

In the area of polynomial time problems, Hartline and Sharp [3] introduce incremental versions of max flow, and find the first instance of a polynomial-time problem whose incremental version is NP-hard. In [7] we present a general model and algorithmic framework for combinatorial covering problems, achieving a 4α -approximation for any incremental covering problem with an α -approximation for its offline version.

Online problems share many similarities with the incremental model. For instance, their input changes with time and their solutions build on each other incrementally. However, online algorithms act with no knowledge of future input and are evaluated only on their final output [8, 9]. The performance of an online algorithm is typically measured against that of the best off-line algorithm. This compares a solution built up over time to the best solution at the very last level. If one is concerned with intermediate solutions, then it is more reasonable to compare the sequence of online solutions with the best incremental solution. In particular, it may not be possible to obtain reasonable solutions at each level while simultaneously guaranteeing a final solution with good competitive ratio. Therefore the competitive ratio may be lower than what can reasonably be expected by an online algorithm attempting to be fair at each time step. Online algorithms have been studied in many contexts, including bin packing [10], graph coloring [11], and bipartite matching [12]. Analysis of these online algorithms could benefit from theoretical results on the corresponding off-line incremental problem. This issue is briefly addressed in [7].

Stochastic optimization problems also bear some resemblance to our incremental framework, in that they have multi-stage input and incremental solutions. However, the problem instance is not fully known at the outset, and the goal is to find a single solution of minimum cost. We motivate our general model by those developed for stochastic problems [13–15]. General models for single-level optimization problems are available in [16, 17].

This large field of related work motivates many possible extensions to the results discussed in this paper. Our incremental model can be extended to handle incomplete knowledge of future constraints, such as with online and stochastic problems. It is worth investigating a model that relaxes the incremental constraint but charges some price for every violation, as seen in on-line bipartite matching [12]. Alternatively, one could relax the packing constraint but charge some price for each broken constraint. Lastly, any given optimization problem has many potential incremental variants. We discuss a few specific problems in this paper. See [3] for results for various incremental versions of max flow.

Table 1. Best known approximation factors for some maximization problems and their incremental variants. *: n is the number of nodes in the flow network. **: fully polytime approximation scheme (FPTAS) from [18, 19]. \mathcal{H}_k is the k^{th} harmonic number, which is $\Theta(\log(k))$

Problem	Single-Level	k -Level Max Sum	k -Level Max Ratio
Bipartite Matching	1	1	$\frac{1}{2}$ ($k = 2$) $\frac{1}{2}$ ($k > 2$)
Weighted Bipartite Matching	1	1	NP-hard ($k \geq 2$)
Max Flow	1	$1/\mathcal{H}_k$ (tight)	$O(1/n)^*$ (tight)
Knapsack	$1 - \epsilon^{**}$	$1/\mathcal{H}_k$	$(1 - \epsilon)^2/2$

2 Preliminaries

Single-Level Problems. We define a single-level abstract combinatorial optimization problem that we adapt to the incremental setting. Such a problem Π consists of a ground set from which we select a subset of elements of optimal value that satisfy input constraints. In particular, let X be the ground set, $\mathcal{F} \subseteq 2^X$ the set of *feasible solutions* as defined by problem constraints, and $v : 2^X \rightarrow \mathbb{R}$ a valuation function on element sets. The goal is to return an $S \in \mathcal{F}$ optimizing $v(S)$. Let $\text{OPT}(X, \mathcal{F}, v)$, or $\text{OPT}(\mathcal{F})$ when X and v are understood, denote such a solution.

This notation is adapted from the general minimization models of [13, 14], however it is general enough to represent both maximization and minimization problems. This paper considers packing problems, a subclass of maximization problems that are “monotone,” in the sense that any subset of a feasible solution is also feasible. In particular, if \mathcal{F} is nonempty then the empty set is a feasible solution: $\emptyset \in \mathcal{F}$. We further assume that $v(\emptyset) = 0$.

Incremental Problems. Given any maximization problem Π , we define its *incremental version* Π^k as follows. There will be k levels. Each level ℓ has its own feasible set \mathcal{F}_ℓ . A feasible solution is a tuple $\mathbf{S} = (S_1, S_2, \dots, S_k)$ such that $S_\ell \in \mathcal{F}_\ell$ and $S_1 \subseteq S_2 \subseteq \dots \subseteq S_k$. Although we do not explicitly assume that $\mathcal{F}_\ell \subseteq \mathcal{F}_{\ell+1}$, we may do so without loss of generality.

In contrast to the single-level problem, where the goal is to find a solution of maximum value, there are several possible objective functions in the incremental variation. For the *maximum ratio* problem, the objective is to satisfy the maximum possible proportion of each level’s optimal solution: find \mathbf{S} maximizing $\mathcal{R}(\mathbf{S}) = \min_\ell \frac{v(S_\ell)}{v(\text{OPT}(\mathcal{F}_\ell))}$. This is the same as the competitive ratio of an online problem, and is a standard metric for incremental problems [1, 2]. If one is less concerned about fairness over all levels, and is more concerned about overall performance, then the maximum sum objective is more appropriate: for the *maximum sum* problem, the objective is to maximize the sum of the solutions over all levels: find \mathbf{S} maximizing $\mathcal{V}(\mathbf{S}) = \sum_\ell v(S_\ell)$.

We now consider three well-known problems, and demonstrate how they fit into this framework. There are multiple ways to define incremental versions of these problems, but we introduce only those subject to discussion in this paper.

2.1 Bipartite Matching

The bipartite matching problem is defined on a graph $G = (U \cup V, E)$; the elements are the edges of the graph, and the feasible solutions are matchings contained in E . The value of a matching M is $v(M) = |M|$.

The incremental version of bipartite matching is defined on a sequence of k bipartite graphs $G_\ell = (U \cup V, E_\ell)$, where $E_\ell \subseteq E_{\ell+1}$. The elements are the edges of E_k , and the feasible set at level ℓ is just the matchings of E_k contained in E_ℓ . Therefore a solution is a sequence of matchings (M_1, M_2, \dots, M_k) such that M_ℓ is a matching in the graph G_ℓ , and $M_\ell \subseteq M_{\ell+1}$. The maximum single-level matching for level ℓ is denoted by M_ℓ^* . The weighted case is defined analogously, except each edge $e \in E_\ell$ has a fixed weight $w_e \geq 0$, and $v(M_\ell) = \sum_{e \in M_\ell} w_e$.

2.2 Maximum Flow

The max flow problem is defined on a directed graph $G = (V, E)$ with source s , sink t , and a capacity function c ; the elements are unit s - t flow paths, and the feasible solutions are the flows satisfying the given capacity function. The value of a flow is the number of unit s - t flow paths it contains.

The incremental version of the max flow problem is defined on a directed graph $G = (V, E)$ with source s , sink t , and a non-decreasing sequence of k capacity functions $c_\ell : E \rightarrow \mathbb{Q}, 1 \leq \ell \leq k$, that define k feasible sets. A solution is a sequence of s - t flows (f_1, f_2, \dots, f_k) such that the flow f_ℓ on any edge e does not exceed the capacity $c_\ell(e)$ but is at least $f_{\ell-1}(e)$, the amount sent along e by the previous flow. We denote the value of a flow f_ℓ by $|f_\ell|$, and the maximum single-level flow at level ℓ by f_ℓ^* .

For other possible interpretations of incremental max flow, see [3].

2.3 Knapsack

The knapsack problem is defined by a knapsack capacity B and a set of items U , item $u \in U$ with size $|u|$ and value v_u ; the elements are the items we could place in our knapsack, while the feasible solutions are subsets of items that fit in the knapsack. In this paper we only consider the case where $v_u = |u|$; the value of a set of items U' is therefore the combined size $|U'| = \sum_{u \in U'} |u|$. This special case is sometimes called the maximum subset sum problem.

The incremental version of knapsack is still defined on a set of items U , item $u \in U$ with size $|u|$, but instead of a single capacity B we have a sequence of k capacities $B_1 \leq B_2 \leq \dots \leq B_k$ that define k feasible sets. A solution is a sequence of subsets (U_1, U_2, \dots, U_k) of U such that $|U_\ell| \leq B_\ell$, and $U_\ell \subseteq U_{\ell+1}$. We denote the value of the maximum single-level solution at level ℓ by B_ℓ^* .

3 The Maximum Sum Objective Function

In this section we discuss how the max sum incremental structure affects the complexity of the problems introduced in Section 2. We give a general technique to convert an algorithm for a problem Π into an approximation algorithm for its incremental variant Π^k , and analyze its tightness with respect to these problems.

Theorem 1. *Max sum weighted incremental bipartite matching is in P .*

Proof. We transform our incremental instance $(G_1, G_2, \dots, G_k, w)$ into a single instance (G, w') of the max weight matching problem, which can then be solved in polytime [20]. We create a graph $G = (V, E)$ where $E = E_k$. For each edge e , we assign it weight $w'_e = w_e \cdot (k - \ell + 1)$ if e first appears in the edge set E_ℓ , i.e. if $e \in E_\ell \setminus E_{\ell-1}$. This is the amount e would contribute to the sum if we were to add it to our solution at level ℓ . For a matching M returned by the max weight matching algorithm, we define an incremental solution $M_\ell = M \cap E_\ell$. We argue that M is a maximum weight matching if and only if (M_1, M_2, \dots, M_k) is the optimal weighted incremental max sum solution. This follows from the one-to-one correspondence between the value of the maximum weight matching and the value of our incremental solution:

$$\begin{aligned} w(M) &= \sum_{e \in M} w'_e = \sum_{\ell=1}^k \sum_{e \in M_\ell \setminus M_{\ell-1}} w'_e = \sum_{\ell=1}^k \sum_{e \in M_\ell \setminus M_{\ell-1}} w_e \cdot (k - \ell + 1) \\ &= \sum_{\ell=1}^k w(M_\ell \setminus M_{\ell-1})(k - \ell + 1) = \sum_{\ell=1}^k w(M_\ell) = \mathcal{V}(M_1, M_2, \dots, M_k). \quad \square \end{aligned}$$

Theorem 1 shows that the max sum incremental structure does not affect the complexity of bipartite matching, suggesting that incremental versions of polytime problems may remain polytime. However, Theorem 3.1 of [3] can be extended to show that adding an incremental structure to max flow alters it enough to significantly change its complexity. This illustrates a dichotomy between the closely related problems of bipartite matching and max flow.

Theorem 2. [3] *The max sum incremental flow problem is NP-hard.*

As there are many incremental problems for which no polytime algorithm exists, we turn our attention to approximation algorithms.

Theorem 3. *Given an α -approximation algorithm ALG for a problem Π , we obtain an $O(\frac{\alpha}{\log k})$ -approximation for its max sum incremental version Π^k .*

Proof. We first run the approximation algorithm for each single-level input to obtain $\text{ALG}(\mathcal{F}_\ell)$ with $v(\text{ALG}(\mathcal{F}_\ell)) \geq \alpha \cdot v(\text{OPT}(\mathcal{F}_\ell))$. We then consider the k incremental solutions

$$\mathbf{H}_\ell = (\underbrace{\emptyset, \emptyset, \dots, \emptyset}_{\ell-1}, \text{ALG}(\mathcal{F}_\ell), \dots, \text{ALG}(\mathcal{F}_\ell))$$

for which $\mathcal{V}(\mathbf{H}_\ell) = (k - \ell + 1) \cdot v(\text{ALG}(\mathcal{F}_\ell))$. Out of these k solutions, return one of maximum value. Denote this solution by \mathbf{H}^* such that for all ℓ

$$\begin{aligned} \mathcal{V}(\mathbf{H}^*) &\geq (k - \ell + 1) \cdot \alpha \cdot v(\text{OPT}(\mathcal{F}_\ell)), \text{ and therefore} \\ v(\text{OPT}(\mathcal{F}_\ell)) &\leq \frac{1}{\alpha} \cdot \frac{1}{k - \ell + 1} \cdot \mathcal{V}(\mathbf{H}^*). \end{aligned}$$

If \mathbf{O}^* is an optimal incremental solution, then

$$\begin{aligned} \mathcal{V}(\mathbf{O}^*) &\leq \sum_{\ell=1}^k v(\text{OPT}(\mathcal{F}_\ell)) \leq \mathcal{V}(\mathbf{H}^*) \cdot \frac{1}{\alpha} \cdot \sum_{\ell=1}^k \frac{1}{k - \ell + 1} \\ &= \mathcal{V}(\mathbf{H}^*) \cdot \frac{\mathcal{H}_k}{\alpha} = \mathcal{V}(\mathbf{H}^*) \cdot O\left(\frac{\log k}{\alpha}\right), \end{aligned}$$

where \mathcal{H}_k is the k^{th} harmonic number. □

While this algorithm is not tight for incremental bipartite matching, Theorem 4 shows it is tight for incremental max flow. The proof relies heavily on gadgetry described in [3], in which we show that any 3-SAT instance can be converted into a two-level unit-capacity directed flow network. This network has two linked components: a clause component c consisting of level 1 edges and a variable component v consisting of level 2 edges. If the clause component appears¹ at level ℓ_c and its corresponding variable component appears at level $\ell_v > \ell_c$, then the results of [3] can easily be extended into the following lemma:

Lemma 1. *Let $\ell'_c \geq \ell_c$ denote the earliest level in which clause component c carries flow. If $\ell'_c < \ell_v$, then any flow through variable component v determines a satisfying assignment. Also, any satisfying assignment can be used to achieve a flow with separate flow paths through components c and v .*

Theorem 4. *Max sum incremental flow is NP-hard to β -approximate, $\beta > \frac{1}{\mathcal{H}_k}$.*

Proof. Suppose we have a $1/(\mathcal{H}_k - \epsilon)$ -approximation for max sum on k -level networks. We solve any instance of 3-SAT by constructing an incremental flow network and using the approximation algorithm to identify satisfiable formulas.

First, let $b = \frac{1}{\epsilon}$. Define $a_0^* = 0$, and $a_\ell^* = \lfloor \frac{bk}{1+k-\ell} \rfloor$ for integers $1 \leq \ell \leq k$. Observe that $\sum_{\ell=1}^k a_\ell^* > bk(\mathcal{H}_k - \epsilon)$ because $\lfloor \frac{bk}{1+k-\ell} \rfloor > \frac{bk}{1+k-\ell} - 1$. Given an instance ϕ of 3-SAT, we build a k -level flow network using $O(b^2 k^2)$ copies of the clause-variable component pairs constructed from ϕ . We create a $b(k-1) \times bk$ matrix of components as shown in Figure 1. Each level ℓ is assigned columns $a_{\ell-1}^* + 1$ through a_ℓ^* . Each such column j contains variable components $v_{1j}, v_{2j}, \dots, v_{a_{\ell-1}^* j}$ and clause components $c_{j(a_\ell^*+1)}, \dots, c_{j(bk-1)}, c_{j(bk)}$, all linked in series between the source and the sink. Components in these columns contain only level ℓ edges. Variable component v_{ab} is linked to clause component c_{ab} .

In this construction, the maximum flow possible at level ℓ has value a_ℓ^* , thus $UB = \sum_{\ell=1}^k a_\ell^*$ is an upper bound on the flow sum. This is strictly larger than $bk(\mathcal{H}_k - \epsilon)$ as noted earlier. Observe that any level ℓ flow must pass through clause components $c_{j(a_\ell^*+1)}, \dots, c_{j(bk)}$ for some column $j \leq a_\ell^*$. If we ever send more than a_ℓ^* units of flow then this extra flow must pass through variable component $v_{jj'}$ for some $a_\ell^* < j' \leq bk$. Thus by Lemma 1 any flow strictly larger than a_ℓ^* that contains positive flow at level ℓ yields a satisfying assignment for ϕ .

If such an assignment exists, we can achieve the flow sum upper bound UB by applying Lemma 1 to send flow through all clause-variable pairs. If no such assignment exists, consider incremental solution (f_1, f_2, \dots, f_k) and take the smallest ℓ such that $|f_k| \leq a_\ell^*$. Because there is no assignment, $|f_1| = \dots = |f_{\ell-1}| = 0$. Also, $|f_\ell| \leq \dots \leq |f_k| \leq a_\ell^*$, and therefore our flow sum $\sum_\ell |f_\ell| \leq (1+k-\ell)a_\ell^* = (1+k-\ell)(\lfloor \frac{bk}{1+k-\ell} \rfloor) \leq bk$. We use our $1/(\mathcal{H}_k - \epsilon)$ -approximation to distinguish between these cases, and therefore determine whether or not ϕ has a satisfying assignment. \square

The standard pseudo-polynomial dynamic programming algorithm for knapsack can be extended to a $O((B_k)^k)$ algorithm for max sum incremental knapsack. We suspect that similar techniques to those of Section 4 will give a max sum approximation polynomial in k with a better ratio than that established in Theorem 3, however this has yet to be proven.

¹ A component is said to *appear* at level ℓ if, in the incremental flow network, all of its edges have capacity 0 prior to level ℓ and capacity 1 for all subsequent levels.

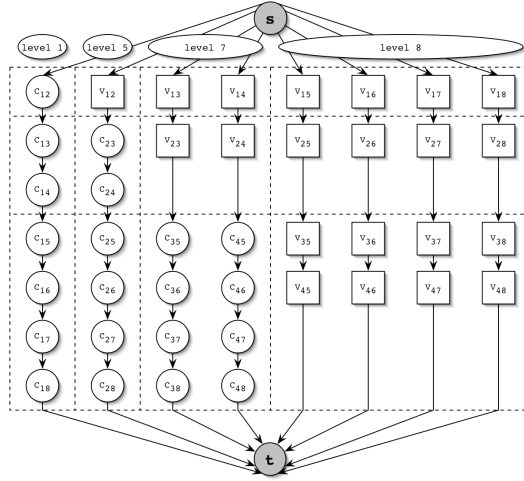


Fig. 1. Circles denote clause components and squares denote variable components. Clause-variable tuple (c_{ij}, v_{ij}) consists of clause c_{ij} component in column i and variable v_{ij} in row j . $k = 8$ and $\epsilon = b = 1$.

4 The Maximum Ratio Objective Function

In this section, we give analogous results for the max ratio objective function.

Theorem 5. *Max ratio 2-level incremental bipartite matching is in P.*

Proof. We transform an incremental instance G_1, G_2 into a single instance (G, w) of the maximum weight matching problem. We create a graph $G = (V, E)$ with $E = E_2$. For each edge e , we assign it weight 1 if $e \in E_1$ and 0 otherwise. For each $1 \leq m \leq |M_2^*|$ we find the max weight matching M^m of size m . From each such matching we define an incremental solution $M_\ell^m = M^m \cap E_\ell$. We return a solution (M_1^m, M_2^m) of maximum ratio. By the nature of the weights given to level 1 edges, if an (m', m) matching exists then $|M_1^m| \geq m'$. Therefore our solution must have a ratio no worse than that of the (m', m) matching. \square

This technique can be generalized for arbitrary k when the optimal ratio r^* is 1. However, the following theorem shows that adding weights makes the problem intractable, and distinguishes it from the polytime max sum version.

Theorem 6. *Max ratio 2-level weighted incremental matching is NP-hard.*

The proof of Theorem 6 follows from a reduction from *partition*, known to be NP-complete [21]. Given a finite set A and sizes $s(a) \in \mathbb{Z}^+$ for all a in A , the partition problem finds a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$. We construct a 2-level instance of weighted incremental bipartite matching such that a ratio of $\frac{1}{2}$ is achievable if and only if the desired partition of A exists.

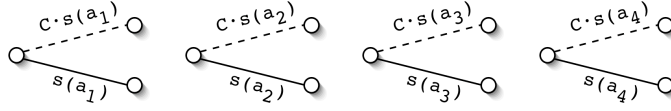


Fig. 2. A weighted incremental bipartite matching instance constructed from an instance of partition with $A = \{a_1, a_2, a_3, a_4\}$. Solid and dashed lines represent level 1 and level 2 edges, respectively. Edges are labeled with their weights.

For each element $a \in A$, we create a 2-level gadget consisting of two incident edges e_a^1 and e_a^2 . Edge e_a^1 is a level 1 edge of weight $s(a)$, and edge e_a^2 is a level 2 edge of weight $C \cdot s(a)$, for some $C > 1$. This construction is shown in Figure 2.

Let $S = \sum_{a \in A} s(a)$. Then the optimal level 1 matching M_1^* selects all level 1 edges, with total weight S . The optimal level 2 matching M_2^* selects all level 2 edges, with total weight $C \cdot S$. Let us define $C = S + 1$.

Lemma 2. *There is a partition of A if and only if there exists an incremental matching achieving ratio $\frac{1}{2}$.*

[\Rightarrow] Suppose we have a partition $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a) = \frac{S}{2}.$$

We create an incremental matching (M_1, M_2) by selecting $M_1 = \{e_a^1 \mid a \in A'\}$ and $M_2 = M_1 \cup \{e_a^2 \mid a \in A \setminus A'\}$. This is a feasible solution, as we use exactly one gadget edge for each element $a \in A$ in our incremental matching. Furthermore,

$$\begin{aligned} r_1 &= \frac{w(M_1)}{w(M_1^*)} & r_2 &= \frac{w(M_2)}{w(M_2^*)} \\ &= \frac{\sum_{a \in A'} s(a)}{S} & &= \frac{w(M_1) + C \sum_{a \in A \setminus A'} s(a)}{C \cdot S} \\ &= \frac{1}{2} & &= \frac{\frac{S}{2} + C \cdot \frac{S}{2}}{C \cdot S} \geq \frac{1}{2} \end{aligned}$$

[\Leftarrow] Now suppose we have an incremental matching (M_1, M_2) achieving ratio $\frac{1}{2}$. First we claim that $w(M_1) = \frac{S}{2}$: If not, then in order to achieve the stated ratio, we have $w(M_1) \geq \frac{S}{2} + 1$, and hence $w(M_2) \leq \frac{S}{2} + 1 + C \cdot (\frac{S}{2} - 1)$. But then

$$\begin{aligned} r_2 &\leq \frac{\frac{S}{2} + 1 + C \cdot (\frac{S}{2} - 1)}{C \cdot S} &= &\frac{1}{2} + \frac{S + 2 - 2C}{2 \cdot C \cdot S} \\ &= \frac{1}{2} + \frac{S + 2 - 2 \cdot (S + 1)}{2 \cdot (S + 1) \cdot S} &= &\frac{1}{2} - \frac{1}{2 \cdot (S + 1)} < \frac{1}{2} \end{aligned}$$

which contradicts our ratio of $\frac{1}{2}$. Therefore we define A' to be the elements a such that $e_a^1 \in M_1$ so that $\sum_{a \in A'} s(a) = w(M_1) = \frac{S}{2}$. \square

Theorem 6 follows directly from Lemma 2. However, the hardness of incremental bipartite matching with polysize weights is unknown, as there is a pseudo-polynomial time dynamic programming algorithm for partition.

Despite the hardness of weighted incremental bipartite matching, Theorem 5 still manages to distinguish incremental matching from incremental max flow, which is NP-hard for $r^* = 1$ and unit capacities. This follows from an extension of the following theorem from [3].

Theorem 7. [3] *The max ratio incremental flow problem is NP-hard.*

Furthermore, [3] proves that the greedy algorithm, which repeatedly sends the maximum level ℓ flow given the incremental constraints imposed by previous levels, is a $\frac{1}{n}$ -approximation algorithm, and this algorithm is tight.

Theorem 8. [3] *Max ratio incremental flow is NP-hard to $g(n)$ -approximate for $g \in \omega(\frac{1}{n})$.*

In summary, we have a complete picture of hardness for max ratio incremental flow, and results for many cases of incremental bipartite matching. However, we have yet to discover the hardness of 3-level incremental bipartite matching with unit weights and $r < 1$, or for weighted bipartite matching with polysize weights. Now we turn our eye to the final problem: we present a constant-factor approximation algorithm for the incremental max ratio knapsack problem.

We introduce some assumptions and notation before we present Lemmas 3-5 and the resulting algorithm. Let r^* denote the optimal max ratio. We assume items $U = \{u_1, u_2, \dots, u_n\}$ are ordered by non-decreasing size, and we define σ_j to be the sum of the sizes of the first j items in this ordering. We say that level ℓ is σ -good if $\frac{r^*}{2}B_\ell^* \leq \sigma_j \leq B_\ell$ for some j , i.e. if the j smallest items are an $\frac{r^*}{2}$ -approximation to B_ℓ^* . Level ℓ is σ -bad if it is not σ -good. If level ℓ is σ -bad then there is some j such that $\sigma_j < \frac{r^*}{2}B_\ell^*$ but $\sigma_{j+1} > B_\ell$. The following lemma, stated without proof, implies that the optimal incremental solution for this level contains an item at least as big as u_{j+1} . We call this item level ℓ 's *required item*.

Lemma 3. *Given knapsack size B and solution \hat{U} , if $U' \subseteq U$ is a maximal solution but $|U'| < |\hat{U}|/2$, then \hat{U} contains exactly one item of size greater than $|\hat{U}|/2$.*

Lemma 4. *If u_j is the required item of the last σ -bad level ℓ , then any $\frac{r^*}{2}$ -approximation for levels $1..l$ with $u_j \in U_\ell$ can be extended to an $\frac{r^*}{2}$ -approximation for levels $1..k$.*

Proof. By definition of ℓ and u_j we have $|u_j| > (1 - \frac{r^*}{2})B_\ell > \frac{1}{2}B_\ell$. Therefore any solution requiring $u_j \in U_\ell$ cannot contain items of size greater than $|u_j|$ in any of the first ℓ levels. Each level $h > \ell$ is σ -good, thereby having some $\sigma_{i_h} \geq \frac{r^*}{2}B_h^* \geq \frac{r^*}{2}B_\ell^*$. As any solution with $u_j \in U_\ell$ only contains items also in σ_{i_h} for all $h > \ell$, and all such levels h are σ -good, we can extend any such solution to all k levels by using the σ_{i_h} solution on levels $h > \ell$. \square

Lemma 5. *If u_j is the required item of the last σ -bad level ℓ , then there exists some level $\ell' \leq \ell$ where we can place u_j such that an r^* solution still exists for levels $1..l' - 1$.*

Proof. Consider some optimal incremental solution, and let ℓ' be the earliest level that uses some item $u_{j'}$ at least as big as u_j . Replacing $u_{j'}$ with u_j in this solution does not affect the ratio achieved for levels 1 through $\ell' - 1$. \square

The dynamic programming solution presented below assumes we know the optimal ratio r^* as well as the optimal single-level solutions $B_1^*, B_2^*, \dots, B_k^*$. Under these assumptions, the algorithm achieves a $\frac{1}{2}$ -approximation to the max ratio knapsack problem. We will remove these assumptions later at the cost of adding a $(1 - \epsilon)^2$ -factor to the approximation bound.

Knapsack Algorithm. Add dummy level B_{k+1} and item u_{n+1} with $B_{k+1} \gg B_k$ and $|u_{n+1}| = |u_n|$. We build a table $M[1..k, 1..n]$. Entry $M[\ell, j]$ is an $\frac{r^*}{2}$ -solution for levels $1.. \ell$, items $\{u_1, u_2, \dots, u_j\}$, and modified capacities $B_i = \min\{B_i, B_{\ell+1} - |u_{j+1}|\}$ if we find a solution and \emptyset otherwise. If an r^* solution exists for this modified problem, we guarantee $M[\ell, j] \neq \emptyset$. We return $M[k, n]$.

$M[0, j]$ is the empty tuple as there are no levels. To compute $M[\ell, j]$ we assume that subproblem $[\ell, j]$ has an r^* solution. If this is not the case then the value of the entry does not matter, and we can set $M[\ell, j] = \emptyset$ if we ever have trouble executing the following procedure.

We first consider the smallest item first solution. If all levels are σ -good we return this greedy $\frac{r^*}{2}$ -solution. Otherwise, there is at least one σ -bad level. Let $u_{j'}$ be the required item of the last σ -bad level y , which must exist assuming an r^* solution is feasible.

We pick the first $1 \leq \ell' \leq y$ such that $B_{\ell'} > u_{j'}$ and $M[\ell' - 1, j' - 1] \neq \emptyset$. We solve levels $1.. \ell' - 1$ using $M[\ell' - 1, j' - 1]$, levels $\ell'..y$ by adding $u_{j'}$ at level ℓ' , and levels $y + 1.. \ell$ with the smallest item first algorithm. Levels $1.. \ell' - 1$ are satisfied by definition of $M[\ell' - 1, j' - 1]$, levels $\ell'..y$ are satisfied by Lemma 3, and levels $y + 1.. \ell$ are satisfied by Lemma 4. Moreover, because an r^* solution is feasible, Lemma 5 guarantees that such an ℓ' exists. If no such ℓ' exists, it must have been because no r^* solution was possible, and we set $M[\ell, j] = \emptyset$.

The running time of this algorithm is dominated by the computation of $M[\ell, j]$ for all nk entries. Each entry requires $O(n)$ time and therefore the running time is $O(kn^2)$.

Theorem 9. *For incremental knapsack, there is a $\frac{(1-\epsilon)^2}{2}$ -approximation to the max ratio objective function that runs in time $O(\frac{k^2 n^5}{\epsilon} \frac{\log(u_n)}{-\log(1-\epsilon)})$.*

Proof. Given r^* and B_ℓ^* for all levels ℓ , the above algorithm $\frac{1}{2}$ -approximates max ratio knapsack. Although determining B_ℓ^* is NP-complete, we can use an FPTAS to find a solution \hat{U}_ℓ with $|\hat{U}_\ell| \geq (1 - \epsilon)B_\ell^*$ in time $O(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$ [18, 19].

The greedy algorithm (smallest item first) gives us a lower bound on r^* of $1/u_n$. We run our algorithm with $r^* = 1/u_n$. If we find a ratio $\frac{r^*}{2}$ solution then multiply r^* by $1/(1 - \epsilon)$ and try again. We continue until the algorithm fails to find a $1/2$ -approximation for some $r^* = \frac{1}{u_n} (\frac{1}{1-\epsilon})^q$. At this point, $\frac{1}{u_n} (\frac{1}{1-\epsilon})^{q-1} \leq r^* < \frac{1}{u_n} (\frac{1}{1-\epsilon})^q$, so if we take $\hat{r} = \frac{1}{u_n} (\frac{1}{1-\epsilon})^{q-1}$ then $\hat{r} \geq (1 - \epsilon)r^*$. This may take $\frac{\log(u_n)}{-\log(1-\epsilon)}$ iterations, but can be accelerated by binary search.

With \hat{r} and \hat{U}_ℓ , the algorithm finds a solution (U_1, U_2, \dots, U_k) such that for each level ℓ

$$|U_\ell| \geq \frac{\hat{r}}{2} |\hat{U}_\ell| \geq \frac{(1-\epsilon)^2}{2} \cdot r^* B_\ell^*.$$

The time needed to compute \hat{r} , \hat{U}_ℓ , and run the algorithm is $O(k^2 n^4 \lfloor \frac{n}{\epsilon} \rfloor \cdot \frac{\log(u_n)}{-\log(1-\epsilon)})$. \square

References

1. Plaxton, C.G.: Approximation algorithms for hierarchical location problems. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, ACM Press (2003) 40–49

2. Mettu, R.R., Plaxton, C.G.: The online median problem. In: FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, Washington, DC, USA, IEEE Computer Society (2000) 339
3. Hartline, J., Sharp, A.: Hierarchical flow. Technical Report 2004-1965, Computer Science Department, Cornell University (2004)
4. Lin, G., Nagarajan, C., Rajaraman, R., Williamson, D.P.: A general approach for incremental approximation and hierarchical clustering. In: SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms. (2006)
5. Fang, Y., Lou, W.: A multipath routing approach for secured data delivery. In: Proceedings of IEEE Micron 2001, IEEE Computer Society (2001) 1467–1473
6. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* **38** (1985) 293–306
7. Hartline, J., Sharp, A.: An incremental model for combinatorial minimization problems. Available at www.cs.cornell.edu/~asharp. (2006)
8. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)
9. Fiat, A., Woeginger, G.J., eds.: Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996). In Fiat, A., Woeginger, G.J., eds.: Online Algorithms. Volume 1442 of Lecture Notes in Computer Science., Springer (1998)
10. Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. *SIAM Journal on Computing* **12** (1983) 227–258
11. Gyarfas, A., Lehel, J.: Online and first-fit colorings of graphs. *J. Graph Th.* **12** (1988) 217–227
12. Karp, R.M., Vazirani, U.V., Vazirani, V.V.: An optimal algorithm for on-line bipartite matching. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing, ACM Press (1990) 352–358
13. Gupta, A., Pal, M., Ravi, R., Sinha, A.: Boosted sampling: approximation algorithms for stochastic optimization. In: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, ACM Press (2004) 417–426
14. Immorlica, N., Karger, D., Minkoff, M., Mirrokni, V.S.: On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2004) 691–700
15. Dean, B.C., Goemans, M.X., Vondrak, J.: Adaptivity and approximation for stochastic packing problems. In: To appear in the 16th annual ACM-SIAM Symposium on Discrete Algorithms. (2005)
16. Plotkin, S.A., Shmoys, D.B., Tardos, E.: Fast approximation algorithms for fractional packing and covering problems. In: Proceedings of the 32nd annual symposium on Foundations of computer science, IEEE Computer Society Press (1991) 495–504
17. Garg, N., Koenemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Proceedings of the 39th Annual Symposium on Foundations of Computer Science, IEEE Computer Society (1998) 300
18. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM* **22** (1975) 463–468
19. Lawler, E.L.: Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research* **4** (1979) 339–356
20. Kuhn, H.: The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* **2** (1955) 83–97

21. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W., eds.: Complexity of Computer Computations. Plenum Press (1972) 85–103