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.

1 Introduction

There has been recent interest in incremental versions of classic problems such as facility location [1], k-median [2], and maximum flow [3]. 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 problem constraints increases at discrete time intervals; we assume these constraints are known in advance. The solution is allowed to change at each time step, but it must always contain the preceding solution as a subsolution. The quality of this multi-stage solution depends on the quality of each intermediate solution. Although we would like a good solution at every stage, a commitment at some stage may limit the options at later stages. It is this tension between local and global optimality that makes incremental problems challenging.

As an example, consider the *incremental bipartite matching* problem 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. One possible application is job scheduling, as it can be costly or disruptive to reassign jobs.

Following this framework, the *incremental flow* problem is defined on a directed network with source s, sink t, and a non-decreasing sequence of capacity functions. 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. Flight scheduling, in which discontinuing a flight leg is undesirable, is one possible application of incremental flow.

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 possible 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 sum and maximum ratio. We find that the incremental version of bipartite matching remains polytime in many cases, whereas the corresponding model of the closely related max flow problem becomes NP-complete. We give a general technique to translate exact or approximate algorithms for non-incremental optimization problems into approximation algorithms for the corresponding incremental versions for both max sum and max ratio objectives. 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 Figure 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; Dasgupta [4] studies an incremental version of the k-center problem, Plaxton [1] and later with Mettu [2] study incremental versions of the uncapacitated k-median and facility location problems, and in [3] we introduce incremental versions of max flow. The problems of [4, 1, 2] do not fit our general model, as they are minimization problems and we deal only with maximization problems in this paper. Several incremental max flow variants under the max ratio objective are studied in [3]. The current paper, on the other hand, explores a model applicable to any maximization problem and gives general results for both max ratio and max sum objectives, as well as specific results for a few additional problems.

On-line problems share many similarities with the incremental model. For instance, their input changes with time and their solutions build on each other incrementally. However, on-line algorithms act with no knowledge of future input and are evaluated only on their final output. The performance of an on-line 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 on-line 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 on-line algorithm attempting to be fair at each time step. On-line algorithms have been studied in many contexts, including bin packing [5], graph coloring [6], and bipartite matching [7]. Analysis of these on-line algorithms could benefit from theoretical results on the corresponding off-line incremental problem.

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 models by those developed for stochastic problems [8–10]. General models for single-level optimization problems are available in [11, 12].

This large field of related work motivates many possible extensions to the results discussed in this paper. We have several results regarding a general model for incremental minimization problems, which will be subject of a future paper. Our incremental model could also be extended to handle incomplete knowledge of future constraints, such as with on-line 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 [13]. Lastly, any given optimization problem has many potential incremental variants. 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 [14, 15]. $\mathcal{H}_k$ 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	1 $(k = 2)$ 1/2 $(k > 2)$
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, $\mathcal{S} \subseteq 2^{\mathtt{X}}$ be the set of *feasible solutions* as defined by problem constraints, and $v : 2^{\mathtt{X}} \to \mathbb{R}$ be a valuation function on element sets. The goal is to return an $F \in \mathcal{S}$ optimizing $v(F)$. Let $\mathsf{OPT}(\mathcal{S})$ denote such a solution.

This notation is adapted from the general minimization models of [8, 9], 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 $\mathcal{S}$ is nonempty then the empty set is a feasible solution: $\emptyset \in \mathcal{S}$. We further assume that $v(\emptyset) = 0$.

Incremental Problems. Given any maximization problem Π, we define its *incremental version* $\mathsf{Inc}_k(\Pi)$ as follows. There will be k *levels*. Each level i has its own feasible set $\mathcal{S}_i$. A feasible solution is a tuple $\mathbf{F} = (F_1, F_2, \ldots, F_k)$ such that $F_i \in \mathcal{S}_i$ and $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_k$. Although we do not explicitly assume that $\mathcal{S}_i \subseteq \mathcal{S}_{i+1}$, 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 sum* problem, the objective is to maximize the sum of the solutions over all levels: find $\mathbf{F}$ maximizing $v_{sum}(\mathbf{F}) = \sum_i v(F_i)$. For the *maximum ratio* problem, the objective is to satisfy the maximum possible proportion of each level's optimal solution: find $\mathbf{F}$ maximizing $v_{ratio}(\mathbf{F}) = \min_i \frac{v(F_i)}{\mathsf{OPT}(\mathcal{S}_i)}$. This is a standard metric for incremental problems [1, 2].

We now consider three well-known problems, and demonstrate how they fit into the framework above. There are various ways to define the incremental versions of these problems, but we will only introduce the ones 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 $G_i = (U \cup V, E_i)$, where $E_i \subseteq E_{i+1}$. The elements are the edges of E_k, and the feasible set at level i is just the matchings of E_k contained in E_i. Therefore a solution is a sequence of matchings $(M_1, M_2, \ldots, M_k)$ such that M_i is a matching in the graph G_i, and $M_i \subseteq M_{i+1}$. The maximum single-level matching for level i is denoted by M_i^*.

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 $c_i : E \to \mathbb{Q}, 1 \leq i \leq k$, that define k feasible sets. A solution is a sequence of s-t flows $(f_1, f_2, \ldots, f_k)$ such that the flow f_i on any edge e does not exceed the capacity $c_i(e)$ but is at least $f_{i-1}(e)$, the amount sent along e by the previous flow. We denote the value of a flow f_i by $|f_i|$, and the maximum single-level flow at level i by f_i^*.

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 $B_1 \leq B_2 \leq \cdots \leq B_k$ that define k feasible sets. A solution is a sequence of subsets $(U_1, U_2, \ldots, U_k)$ of U such that $|U_i| \leq B_i$, and $U_i \subseteq U_{i+1}$. We denote the value of the maximum single-level solution at level i by B_i^*.

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 $\mathsf{Inc}_k(\Pi)$, and analyze its tightness with respect to these problems.

Theorem 1. *The max sum incremental bipartite matching problem is in P.*

Proof. We transform our incremental instance $G_1, G_2, \ldots, G_k$ into a single instance (G, w) of the max weight matching problem, which can then be solved in polytime [16]. We create a graph $G = (V, E)$ where $E = E_k$. For each edge e, we assign it weight $w_e = k - i + 1$ if e first appears in the edge set E_i, i.e. if $e \in E_i \backslash E_{i-1}$. This is the amount e would contribute to the sum if we were to add it to our solution at level i. For a matching M returned by the max weight matching algorithm, we define an incremental solution $M_i = M \cap E_i$. We argue that M is a maximum weight matching if and only if $(M_1, M_2, \ldots, M_k)$ is the optimal 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:

$$w(M) = \sum_{e \in M} w(e) \;=\; \sum_{i=1}^{k} \sum_{e \in M_i \setminus M_{i-1}} w(e) \;=\; \sum_{i=1}^{k} \sum_{e \in M_i \setminus M_{i-1}} (k-i+1)$$
$$= \sum_{i=1}^{k} |M_i \setminus M_{i-1}|(k-i+1) \;=\; \sum_{i=1}^{k} |M_i| \;=\; v_{sum}(M_1, M_2, \ldots, M_k). \quad \square$$

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, the proof of 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, illustrating 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.*

In addition to incremental flow, and due to the fact that incremental variants of NP-complete problems contain their single-level variants as special cases, there are potentially many incremental problems for which no exact algorithm exists. We therefore turn our attention to obtaining approximation algorithms for the max sum objective function.

Theorem 3. *Given an α-approximation algorithm ALG for a single-level problem Π, we obtain an $O(\frac{\alpha}{\log k})$-approximation for the max sum incremental version $\mathsf{Inc}_k(\Pi)$.*

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

$$\mathbf{H_i} = (\underbrace{\emptyset, \emptyset, \ldots, \emptyset}_{i-1}, \mathsf{ALG}(\mathcal{S}_i), \ldots, \mathsf{ALG}(\mathcal{S}_i))$$

for which $v_{sum}(\mathbf{H_i}) = (k-i+1) \cdot v(\mathsf{ALG}(\mathcal{S}_i))$. Out of these k solutions, return one of maximum value. Denote this solution by $\mathbf{H}^*$ so that for all i

$$v_{sum}(\mathbf{H}^*) \geq (k-i+1) \cdot \alpha \cdot v(\mathsf{OPT}(\mathcal{S}_i)), \text{ and therefore}$$
$$v(\mathsf{OPT}(\mathcal{S}_i)) \leq \frac{1}{\alpha} \cdot \frac{1}{k-i+1} \cdot v_{sum}(\mathbf{H}^*).$$

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

$$v_{sum}(\mathbf{O}^*) \leq \sum_{i=1}^{k} v(\mathsf{OPT}(\mathcal{S}_i)) \;\leq\; v_{sum}(\mathbf{H}^*) \cdot \frac{1}{\alpha} \cdot \sum_{i=1}^{k} \frac{1}{k-i+1}$$
$$= v_{sum}(\mathbf{H}^*) \cdot \frac{\mathcal{H}_k}{\alpha} \quad = \quad v_{sum}(\mathbf{H}^*) \cdot O\left(\frac{\log k}{\alpha}\right),$$

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

While this algorithm is not tight for bipartite matching, Theorem 4 shows it is tight for 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 incremental flow network. This network contains two linked components: a clause component c and a variable component v. If the clause component appears[1] 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 to prove the following lemma:

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

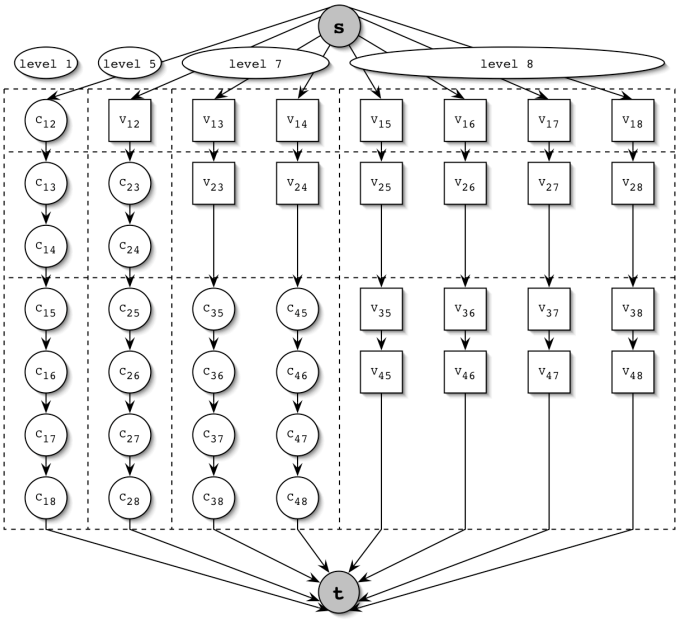

Fig. 1. Circles denote clause components and squares denote variable components. Clause-variable tuple (c_{ij}, v_{ij}) consists of clause c_{ij} component in column i and variable v_{ij} in row i. $k = 8$ and $\epsilon = b = 1$.

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

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

Proof. Suppose we are given a $1/(\mathcal{H}_k - \epsilon)$-approximation algorithm for max sum on k-level flow networks. Then 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 $a_i^* = \lfloor \frac{bk}{1+k-i} \rfloor$ for integers $1 \leq i \leq k$. Observe that $\sum_{i=1}^{k} a_i^* > bk(\mathcal{H}_k - \epsilon)$ because $\lfloor \frac{bk}{1+k-i} \rfloor > \frac{bk}{1+k-i} - 1$. Given an instance ϕ of 3-SAT, we build a k-level flow network using $O(b^2k^2)$ 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 i is assigned columns $a_{i-1}^* + 1$ through a_i^*. Each such column j contains variable components $v_{1j}, v_{2j}, \ldots, v_{a_{i-1}^* j}$ and clause components $c_{j(a_i^*+1)}, \ldots, c_{j(bk-1)}, c_{j(bk)}$, all linked in series between the source and the sink. Components in these columns contain only level i edges. Variable component v_{ab} is linked to clause component c_{ab}.

In this construction, the maximum flow possible at level i has value a_i^*. An upper bound on the flow sum over all levels is thus $UB = \sum_{i=1}^{k} a_i^*$, which is strictly larger than $bk(\mathcal{H}_k - \epsilon)$ as noted earlier. Observe that any level i flow must pass through clause components $c_{j(a_i^*+1)}, \ldots, c_{j(bk)}$ for some column $j \leq a_i^*$. If we ever send more than a_i^* units of flow then this extra flow must pass through variable component $v_{jj'}$ for some $a_i^* < j' \leq bk$. Thus by Lemma 1 any flow strictly larger than a_i^* that contains positive flow at level i 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 $(f_1, f_2, \ldots, f_k)$ and take the smallest i such that $|f_k| \leq a_i^*$. Because there is no assignment, $|f_1| = \ldots = |f_{i-1}| = 0$. Also, $|f_i| \leq \ldots \leq |f_k| \leq a_i^*$, and therefore our flow sum $\sum_i |f_i| \leq (1+k-i)a_i^* = (1+k-i)(\lfloor \frac{bk}{1+k-i} \rfloor) \leq bk$. 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. □

The standard pseudo-polynomial dynamic programming algorithm for knapsack can be extended to a $O((B_k)^k)$ algorithm for max sum incremental knapsack. Furthermore, we suspect that the techniques described in Section 4 can be used to give a max sum approximation polynomial in k with a better ratio than the lower bound established in Theorem 3.

4 The Maximum Ratio Objective Function

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

Theorem 5. *The max ratio 2-level incremental bipartite matching problem 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 $1 \leq j \leq |M_2^*|$ we find the max weight matching M^j of size j. From each such matching we define an incremental solution $M_i^j = M^j \cap E_i$. We return a solution (M_1^j, M_2^j) of maximum ratio. By the nature of the weights given to level 1 edges, if an (j', j) matching exists then $|M_1^j| \geq j'$. Therefore our solution must have a ratio no worse than that of the (j', j) matching. □

This technique can be generalized to an arbitrary number of levels for optimal ratio $r^* = 1$, but not in general. Nevertheless, Theorem 5 distinguishes incremental matching from incremental max flow, which is NP-hard for $r^* = 1$. This follows from an extension of the following theorem from [3].

Theorem 6. *[3] The max ratio incremental flow problem is NP-hard.*

We now consider approximation algorithms for the max ratio incremental model. For an incremental problem $\mathsf{Inc}_k(\Pi)$ with element set X and feasible solutions $\mathbf{S} = \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$, consider $X' \subseteq X$. Let $\mathsf{OPT}(\mathcal{S}_i|X')$ denote the optimal level i solution containing X' if any such solution exists. For level i, $\frac{v(\mathsf{OPT}(\mathcal{S}_i|X'))}{v(\mathsf{OPT}(\mathcal{S}_i|\emptyset))}$ is the largest ratio of the optimal value we can achieve with a solution forced to contain X'. Define $g(X, \mathbf{S}) = \min\limits_{i, X' \in \mathcal{S}_i} \frac{v(\mathsf{OPT}(\mathcal{S}_i|X'))}{v(\mathsf{OPT}(\mathcal{S}_i|\emptyset))}$ as the worst such ratio for $\mathsf{Inc}_k(\Pi)$.

Theorem 7. *Given an α-approximation to a single-level problem Π, we obtain an $O(\alpha \cdot g(X, \boldsymbol{S}))$-approximation to $\mathsf{Inc}_k(\Pi)$ under the max ratio objective.*

Proof. Consider the greedy algorithm that starts at level 1 with an α-approximate solution, then sequentially finds the best α-approximate solution on each level given the incremental constraint imposed by the previous level's solution. Let F_0 be the empty set and $(F_1, \ldots, F_k)$ the solution we obtain with the above algorithm. Note that $v(\mathsf{OPT}(\mathcal{S}_i)) = v(\mathsf{OPT}(\mathcal{S}_i|\emptyset))$. For $1 \leq i \leq k$,

$$\begin{aligned} v(\mathsf{OPT}(\mathcal{S}_i|F_{i-1})) &\geq g(X, \mathbf{S}) \cdot v(\mathsf{OPT}(\mathcal{S}_i)) \\ v(F_i) \geq \alpha \cdot v(\mathsf{OPT}(\mathcal{S}_i|F_{i-1})) &\ \geq\ g(X, \mathbf{S}) \cdot \alpha \cdot v(\mathsf{OPT}(\mathcal{S}_i)) \end{aligned}$$

Therefore we achieve a ratio of at least $\alpha \cdot g(X, \mathbf{S})$ for each level. □

Corollary 1. *For the max ratio objective function, we have*

i. *a $\frac{1}{2}$-approximation for incremental bipartite matching,*
ii. *a $\frac{1}{n}$-approximation for incremental max flow where $n = |V|$, and*
iii. *a $\frac{1-\epsilon}{u_{max}}$-approximation for incremental knapsack, where $u_{max} = \max_{u \in U} |u|$.*

Proof. For matching, we have that $g(X, \mathbf{S}) = 1/2$; taking edge $e = (u, v)$ at one level blocks at most one other edge from being used by the optimal solution at any level. The max flow result is derived in [3]. For knapsack, an item of size 1 on one level may prevent the largest item (of size u_{max}) from fitting at some higher level, and the best known polytime solution to the single-level knapsack problem is a $(1 - \epsilon)$-approximation [14, 15]. □

It is shown in [3] that this algorithm is tight for max flow.

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

In summary, the general bound is not tight for some cases of bipartite matching, but open for $k > 2$ and $r < 1$. The bound is tight for max flow. Although the single-level version of knapsack is harder than the single-level version of max flow, it turns out that the approximation given by Theorem 7 is not tight for knapsack. We present a constant-factor approximation algorithm for the incremental max ratio knapsack problem.

We introduce some assumptions and notation before we present Lemmas 2-4 and the resulting algorithm. Let r^* denote the optimal max ratio. We assume items $U = \{u_1, u_2, \ldots, u_n\}$ are ordered by non-decreasing size, and we define σ_j to be the sum of the first j item sizes 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, which we state 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 2. *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 3. *If u_j is the required item of the last σ-bad level ℓ, then any $\frac{r^*}{2}$-approximation for levels $1..\ell$ 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 $\sigma_{i_h} \geq \frac{r^*}{2}B^*_h$. 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 $h > \ell$. □

Lemma 4. *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 $1..\ell' - 1$.*

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 $\ell' - 1$. □

The dynamic programming solution presented below assumes we know the optimal ratio r^* as well as the optimal single-level solutions $B^*_1, B^*_2, \ldots, B^*_k$. 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 $B_{k+1} >> B_k$ 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 $\{u_1, u_2 \ldots, u_j\}$, 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 2, and levels $y+1..\ell$ are satisfied by Lemma 3. Moreover, because an r^* solution is feasible, Lemma 4 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 $O(\frac{k^2 n^5}{\epsilon} \frac{\log(u_{max})}{-\log(1-\epsilon)})$.*

Proof. Given r^* and B_i^* for all levels i, the above algorithm $\frac{1}{2}$-approximates max ratio knapsack. Although determining B_i^* is NP-complete, we can use an FPTAS to find a solution $\hat{U}_i$ with $|\hat{U}_i| \geq (1-\epsilon)B_i^*$ in time $O(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$ [14, 15].

Corollary 1 gives us a lower bound on r^* of $1/u_{max}$. We run our algorithm with $r^* = 1/u_{max}$. 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 $r^* = \frac{1}{u_{max}}(\frac{1}{1-\epsilon})^q$. At this point, $\frac{1}{u_{max}}(\frac{1}{1-\epsilon})^{q-1} \leq r^* < \frac{1}{u_{max}}(\frac{1}{1-\epsilon})^q$, so if we take $\hat{r} = \frac{1}{u_{max}}(\frac{1}{1-\epsilon})^{q-1}$ then $\hat{r} \geq (1-\epsilon)r^*$. This may take $\frac{\log(u_{max})}{-\log(1-\epsilon)}$ iterations, but can be accelerated by binary search.

With $\hat{r}$ and $\hat{U}_i$, the algorithm finds a solution $(U_1, U_2, \ldots, U_k)$ such that for each level i

$$|U_i| \;\geq\; \frac{\hat{r}}{2}|\hat{U}_i| \;\geq\; \frac{(1-\epsilon)^{2\cdot}}{2} \cdot r^* B_i^*.$$

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

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, IEEE Computer Society (2000) 339
3. Hartline, J., Sharp, A.: Hierarchical flow. Technical Report 2004-1965, Computer Science Department, Cornell University (2004) Available at http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cis/TR2004-1965.
4. Dasgupta, S.: Performance guarantees for hierarchical clustering. In: Proceedings of the 15th Annual Conference on Computational Learning Theory, Springer-Verlag (2002) 351–363
5. Coffman, E.G., Garey, M.R., Johnson, D.S.: Dynamic bin packing. SIAM Journal on Computing **12** (1983) 227–258
6. Gyárfás, A., Lehel, J.: Online and first-fit colorings of graphs. J. Graph Th. **12** (1988) 217–227
7. 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
8. Gupta, A., Pál, 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
9. 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
10. 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)
11. Plotkin, S.A., Shmoys, D.B., Tardos, É.: 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
12. 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
13. Grove, E.F., Kao, M.Y., Krishnan, P., Vitter, J.S.: Online perfect matching and mobile computing. In: Proceedings of the Workshop on Algorithms and Data Structures. (1995)
14. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM **22** (1975) 463–468
15. Lawler, E.L.: Fast approximation algorithms for knapsack problems. Mathematics of Operations Research **4** (1979) 339–356
16. Kuhn, H.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly **2** (1955) 83–97