

# Greedy in Approximation Algorithms<sup>\*</sup>

Julián Mestre

Department of Computer Science.  
University of Maryland, College Park, MD 20742.

**Abstract.** The objective of this paper is to characterize classes of problems for which a greedy algorithm finds solutions provably close to optimum. To that end, we introduce the notion of  $k$ -extendible systems, a natural generalization of matroids, and show that a greedy algorithm is a  $\frac{1}{k}$ -factor approximation for these systems. Many seemingly unrelated problems fit in our framework, e.g.:  $b$ -matching, maximum profit scheduling and maximum asymmetric TSP.

In the second half of the paper we focus on the maximum weight  $b$ -matching problem. The problem forms a 2-extendible system, so greedy gives us a  $\frac{1}{2}$ -factor solution which runs in  $O(m \log n)$  time. We improve this by providing two linear time approximation algorithms for the problem: a  $\frac{1}{2}$ -factor algorithm that runs in  $O(bm)$  time, and a  $(\frac{2}{3} - \epsilon)$ -factor algorithm which runs in expected  $O(bm \log \frac{1}{\epsilon})$  time.

## 1 Introduction

Perhaps the most natural first attempt at solving any combinatorial optimization problem is to design a greedy algorithm. The underlying idea is simple: we make locally optimal choices hoping that this will lead us to a globally optimal solution. Needless to say that such an algorithm may not always work, therefore a natural question to ask is: for which class of problems does this approach work? A classical theorem due to Edmonds and Rado answers this question; to state this result we first need to define our problem more rigorously.

A *subset system* is a pair  $(E, \mathcal{L})$ , where  $E$  is a finite set of elements and  $\mathcal{L}$  is a collection of subsets of  $E$  such that if  $A \in \mathcal{L}$  and  $A' \subseteq A$  then  $A' \in \mathcal{L}$ . Sets in  $\mathcal{L}$  are called *independent*, and should be regarded as feasible solutions of our problem. Given a positive weight function  $w : E \rightarrow \mathbb{R}^+$  there is a natural optimization problem associated with  $(E, \mathcal{L})$  and  $w$ , namely that of finding an independent set of maximum weight. We want to study the following algorithm, which from now on we simply refer to as Greedy: start from the empty solution and process the elements in decreasing weight order, add an element to the current solution only if its addition preserves independence.

A matroid is a subset system  $(E, \mathcal{L})$  for which the following property holds:

$$\forall A, B \in \mathcal{L} \text{ and } |A| < |B| \text{ then } \exists z \in B \setminus A \text{ such that } A + z^1 \in \mathcal{L}$$

Matroids were first introduced by Whitney [26] as an abstraction of the notion of independence from linear algebra and graph theory. Rado [23] showed that if a given problem has the matroid property then Greedy always finds an optimal solution. In turn, Edmonds [13] proved the other direction of the implication, i.e., if Greedy finds an optimal solution for *any* weight function defined on the elements then the problem must have the matroid property.

A rich theory of matroids exists, see [24, 20] for a thorough treatment of the subject. Many generalizations along two main directions have been proposed. One approach is to define a more general class of problems. Greedy no longer works, therefore alternative algorithms must

---

<sup>\*</sup> Research supported by NSF Awards CCR-01-05413 and CCF-04-30650, and the University of Maryland Dean's Dissertation Fellowship.

<sup>1</sup> The notation  $A + z$  means  $A \cup \{z\}$ , likewise  $A - z$  means  $A \setminus \{z\}$ .

be designed; examples of this are greedoids [17], two-matroid intersection [12], and matroid matching [19]. Another approach is to study structures where Greedy finds optimal solutions for some, but not all weight functions; symmetric matroids [8], symplectic matroids [7] and the work of Vince [25] are along these lines.

Although different in nature, both approaches have the same objective in mind: exact solutions. In this paper we study Greedy from the point of view of approximation algorithms. Our main contribution is the introduction of  $k$ -extendible systems, a natural generalization of matroids. We show that Greedy is a  $\frac{1}{k}$ -factor approximation for  $k$ -extendible systems.

Given a subset system  $(E, \mathcal{L})$ , Korte and Hausman [16] showed that for the maximization problem defined by  $(E, \mathcal{L})$ , Greedy achieves its worst approximation ratio on 0-1 weight functions. Consider the 0-1 function  $w_A$  defined as  $w_A(x) = 1$  for  $x \in A$  and 0 otherwise. The cost of the solution Greedy finds, comes from the elements in  $A$  the algorithm happens to pick, these elements form an independent set which is maximal with respect to  $A$ . Let  $\gamma_A$  be the ratio between the smallest and the largest maximal independent subsets of  $A$ . Notice that  $\gamma_A$  is the worst greedy can do on  $w_A$ . Let  $\gamma = \min_{A \subseteq E} \gamma_A$ . Korte and Hausman showed that Greedy is a  $\gamma$ -factor approximation for  $(E, \mathcal{L})$ .

While this result tells us how well Greedy performs on a particular system, in some cases it may be difficult to establish  $\gamma$  for a given combinatorial problem—which can be regarded as a class of systems, as every instance of the problem defines a system. Our  $k$ -extendible framework better highlights the structure of the problem and allows us to easily explain the performance of Greedy on seemingly unrelated problems such as  $b$ -matching, maximum profit scheduling and maximum asymmetric TSP. For some of these, an algorithm tailored to the specific problem yields a better approximation ratio than that offered by Greedy. This should not come as a surprise, after all Greedy is a generic algorithm that we can try on nearly every problem. The goal of this paper is to characterize those problems for which a simple greedy strategy produces nearly optimal solutions and to better understand its shortcomings. Along these lines is the recent work by Borodin et al. [6], who introduced the paradigm of priority algorithms, a formal class of algorithms that captures most greedy-like algorithms. Lower bounds on the approximation ratio any priority algorithm can achieve were derived for scheduling [6], set cover, and facility location problems [1].

In particular, our framework explains why Greedy produces  $\frac{1}{2}$ -approximate solutions for  $b$ -matching. Given a graph  $G = (V, E)$  with  $n$  vertices and  $m$  edges and degree constraints  $b : V \rightarrow \mathbb{N}$  for the vertices, a  $b$ -matching is a set of edges  $M$  such that for all  $v \in V$  the number of edges in  $M$  incident to  $v$ , denoted by  $\deg_M(v)$ , is at most  $b(v)$ . Polynomial time algorithms exist to solve the problem optimally: A maximum size  $b$ -matching can be found in  $O(nm \log n)$  time and maximum weight in  $O(\sum b(v) \min(m \log n, n^2))$  time; both results are due to Gabow [14]. Greedy on the other hand produces approximate solutions but has the advantage of being simple and much faster, running in just  $O(m \log n)$  time. This time savings can be further improved. For instance, for maximum weight matching (the case where  $b(v) = 1$  for all  $v$ ) Preis [22] proposed a  $\frac{1}{2}$ -approximation algorithm which runs in linear time. Drake et al. [11] designed an alternative simpler algorithm that greedily finds disjoint heavy paths and keeps the best of the two matchings defined on the path; the same authors in later work [10] designed an algorithm with an approximation factor of  $\frac{2}{3} - \epsilon$  which runs in  $O(\frac{m}{\epsilon})$  time. Finally, Pettie and Sanders [21] gave randomized and deterministic algorithms with the same approximation guarantee of  $\frac{2}{3} - \epsilon$  which run in  $O(m \log \frac{1}{\epsilon})$  time. We note that a better approximation ratio can be obtained using local search [2] or the limited-backtrack greedy scheme of Arora *et al* [3], albeit at a very high running time. The challenge here is to get a fast algorithm with a good approximation guarantee.

In the second half of the paper we explore this tradeoff for  $b$ -matching and provide a  $\frac{1}{2}$ -approximation which runs in  $O(bm)$  time and a  $(\frac{2}{3} - \epsilon)$ -factor randomized algorithm that runs in expected  $O(bm \log \frac{1}{\epsilon})$  time, where  $b = \max_u b(u)$ . Our algorithms build upon the work of [11] and [21]. The main difficulty in extending previous results to  $b$ -matching is the way the

optimal solution and the one produced by the algorithm are compared in the analysis. This was done by taking the symmetric difference of the two, which for matchings yields a collection of simple paths and cycles. Unfortunately this does not work for  $b$ -matching, a more careful pairing argument must be provided.

## 2 $k$ -extendible systems

The following definitions are with respect to a given system  $(E, \mathcal{L})$  and a particular weight function. Let  $A \in \mathcal{L}$ , we say  $B$  is an *extension* of  $A$  if  $A \subseteq B$  and  $B \in \mathcal{L}$ . We denote by  $\text{OPT}(A)$  an extension of  $A$  with maximum weight. Note that  $\text{OPT}(\emptyset)$  is an independent set with maximum weight.

**Definition 1.** *The subset system  $(E, \mathcal{L})$  is  $k$ -extendible if for all  $C \in \mathcal{L}$  and  $x \notin C$  such that  $C + x \in \mathcal{L}$  and for every extension  $D$  of  $C$  there exists a subset  $Y \subseteq D \setminus C$  with  $|Y| \leq k$  such that  $D \setminus Y + x \in \mathcal{L}$ .*

Notice that if  $x \in D$  or  $C = D$  then the property holds trivially by letting  $Y = \emptyset$ , therefore we do not need to consider these two cases in our proofs.

Our goal is to characterize problems for which a greedy algorithm will produce good solutions. In Section 2.1 we show that Greedy is a  $\frac{1}{k}$ -factor approximation for  $k$ -extendible systems. We also show a close relation between  $k$ -extendible systems and matroids, starting with the following theorem:

**Theorem 1.** *The system  $(E, \mathcal{L})$  is a matroid if and only if is 1-extendible.*

*Proof.* First we prove the  $\Rightarrow$  direction: given sets  $C \subset D \in \mathcal{L}$  and an element  $x \notin D$  we need to find  $Y$  such that  $D \setminus Y + x$  is independent. Set  $A = C + x$  and  $B = D$ . If  $|A| = |B|$  then the two sets differ by one element, by setting  $Y = B \setminus A$  we get the  $k$ -extendible property. Otherwise we can repeatedly apply the matroid property to add an element from  $B \setminus A$  to  $A$  until  $|A| = |B|$ . Again  $Y = D \setminus A$  has cardinality 1. Since  $D \setminus Y + x = A \in \mathcal{L}$  we get that  $(E, \mathcal{L})$  is 1-extendible.

Let us show the other direction. Given two independent sets  $A$  and  $B$  such that  $|A| < |B|$ , we need to find  $z$ . Notice that if  $A \subseteq B$  we are done, any  $z \in B \setminus A$  will do, this is because any subset of  $B \in \mathcal{L}$  is independent, in particular  $A + z$ . Suppose then that  $A \not\subseteq B$ . The idea is to pick  $x \in A \setminus B$  and then find, if needed, an element  $y$  in  $B \setminus A$  such that  $B - y + x \in \mathcal{L}$ . Remove  $y$  from  $B$ , add  $x$ , and repeat until  $A \subseteq B$ , at this point return any element  $z \in B \setminus A$ .

Pick any  $x$  in  $A \setminus B$ , if  $B + x \in \mathcal{L}$  we are done since we do not need to pick a  $y$ . Otherwise, set  $C = A \cap B$  and  $D = B$ , since the system is 1-extendible there exists  $Y$  such that  $D \setminus Y + x \in \mathcal{L}$ . Moreover  $Y$  consists of exactly one element  $y \in D \setminus C = B \setminus A$ , which is exactly what we were looking for.  $\square$

### 2.1 Greedy

Given  $(E, \mathcal{L})$  and  $w : E \rightarrow \mathbb{R}^+$  a natural first attempt at finding a maximum weight independent set is to use the greedy algorithm on the right. Starting from an empty solution  $S$ , we try to add elements to  $S$  one at a time, in decreasing weight order. We add  $x$  to  $S$  only if  $S + x$  is independent.

```

GREEDY( $G, w$ )
1  sort elements in decreasing weight
2   $S \leftarrow \emptyset$ 
3  for  $x \in E$  in order
4  do if  $S + x \in \mathcal{L}$ 
5     then  $S \leftarrow S + x$ 
6  return  $S$ 

```

**Corollary 1.** *Greedy solves the optimization problem defined by  $(E, \mathcal{L})$  for any weight function if and only if  $(E, \mathcal{L})$  is 1-extendible.*

This follows from Theorem 1 and the work of Rado [23] and Edmonds [13]. Now we generalize one direction of this result for arbitrary  $k$ .

**Theorem 2.** *Let  $(E, \mathcal{L})$  be  $k$ -extendible, Greedy is a  $\frac{1}{k}$ -factor approximation for the optimization problem defined by  $(E, \mathcal{L})$  and any weight function  $w$ .*

Let  $x_1, x_2, \dots, x_l$  be the elements picked by greedy, also let  $S_0 = \emptyset, \dots, S_l$  be the successive solutions, that is  $S_i = S_{i-1} + x_i$ . To prove Theorem 2 we need the following lemma whose proof we defer for a moment.

**Lemma 1.** *If  $(E, \mathcal{L})$  is  $k$ -extendible then the  $i$ th element  $x_i$  picked by Greedy is such that  $w(\text{OPT}(S_{i-1})) \leq w(\text{OPT}(S_i)) + (k-1)w(x_i)$ .*

Remember that we can express the optimal solution as  $\text{OPT}(\emptyset)$ . Starting from  $S_0$  we can apply Lemma 1  $l$  times to get:

$$\begin{aligned} w(\text{OPT}(S_0)) &\leq w(\text{OPT}(S_l)) + (k-1) \sum_{i=1}^l w(x_i) \\ &= w(S_l) + (k-1)w(S_l) \\ &= k w(S_l). \end{aligned}$$

We can replace  $w(\text{OPT}(S_l))$  with  $w(S_l)$  because the set  $S_l$  is maximal. Hence Greedy returns a solution  $S_l$  with cost at least  $\frac{1}{k}$  that of the optimal solution. Now it all boils down to proving Lemma 1.

Notice that  $\text{OPT}(S_{i-1})$  is an extension of  $S_{i-1}$ . Since  $S_{i-1} + x_i \in \mathcal{L}$ , we can find  $Y \subseteq \text{OPT}(S_{i-1}) \setminus S_{i-1}$  such that  $\text{OPT}(S_{i-1}) \setminus Y + x_i \in \mathcal{L}$ . Thus,

$$\begin{aligned} w(\text{OPT}(S_{i-1})) &= w(\text{OPT}(S_{i-1}) \setminus Y + x_i) + w(Y) - w(x_i), \\ &\leq w(\text{OPT}(S_i)) + w(Y) - w(x_i). \end{aligned}$$

The second line follows because  $\text{OPT}(S_{i-1}) \setminus Y + x_i$  is an extension of  $S_{i-1} + x_i$  and  $\text{OPT}(S_i)$  is one with maximum weight. Now let us look at an element  $y \in Y$ , we claim that  $w(y) \leq w(x_i)$ . Suppose for the sake of contradiction that  $w(y) > w(x_i)$ . Since  $y \notin S_{i-1}$  this means that  $y$  was considered by Greedy before  $x_i$  and was dropped. Therefore there exist  $j \leq i$  such that  $S_j + y \notin \mathcal{L}$ , but  $S_j + y \subseteq \text{OPT}(S_{i-1}) \in \mathcal{L}$ , a contradiction. All weights are positive, therefore  $w(Y) \leq kw(x_i)$ , and the lemma follows.

## 2.2 Examples of $k$ -extendible systems

Now we show that many natural problems fall in our  $k$ -extendible framework.

**Maximum weight  $b$ -matching:** Given a graph  $G = (V, E)$  and degree constraints  $b : V \rightarrow \mathbb{N}$  for the vertices, a  $b$ -matching is a set of edges  $M$  such that for all  $v \in V$  the number of edges in  $M$  incident to  $v$ , denoted by  $\deg_M(v)$ , is at most  $b(v)$ .

**Theorem 3.** *The subset system associated with  $b$ -matching is 2-extendible.*

*Proof.* Let  $C + (u, v)$  and  $D$  be valid solutions, where  $C \subseteq D$  and  $(u, v) \notin D$ . We know that  $\deg_C(u) < b(u)$  and  $\deg_C(v) < b(v)$ , otherwise  $C + (u, v)$  would not be a valid solution. Now if  $\deg_D(u) = b(u)$  we can find an edge in  $D \setminus C$  incident to  $u$ , add this edge to  $Y$  and do the same for the other endpoint. Clearly  $D \setminus Y + (u, v) \in \mathcal{L}$  and  $|Y| \leq 2$ , therefore the system is 2-extendible.  $\square$

**Maximum profit scheduling:** We are to schedule  $n$  jobs on a single machine. Each job  $i$  has release time  $r_i$ , deadline  $d_i$ , and profit  $w_i$ , all positive integers. Every job takes the same amount of time  $L \in \mathbb{Z}^+$  to process. (See [4, 9] for an exact algorithm and [5] for a 2-approximation algorithm when the job lengths are arbitrary.) Our objective is to find a non-preemptive schedule that maximizes the weight of the jobs done on time. A job  $i$  is done on time if it starts and finishes in the interval  $[r_i, d_i]$ .

**Theorem 4.** *The subset system associated with maximum profit scheduling is 1-extendible when  $L = 1$ .*

*Proof.* Let  $C + i$  be a feasible set of jobs, and  $D$  an extension of  $C$ . A schedule for a certain set of jobs can be regarded as matching between those jobs and time slots. Let  $M_1$  and  $M_2$  be the matchings for  $C + i$  and  $D$  respectively. The set  $M_1 \cup M_2$  contains a path starting on  $i$  ending on a job  $j \in D \setminus C$  or a time slot  $t$ . Alternating the edges of  $M_2$  along the path we get a schedule for  $D + i - j$  in the first case, and for  $D + i$  in the latter.  $\square$

For  $L > 1$  we model the problem with a slightly different subset system. Let the elements of  $E$  be pairs  $(i, t)$  where  $t$  denotes the time job  $i$  is scheduled, and  $r_i \leq t \leq d_i - L$ . A set of elements is independent if it specifies a feasible schedule. Greedy considers the jobs in decreasing weight and adds the job being processed *somewhere* in the current schedule, if no place is available the job is dropped.

**Theorem 5.** *The subset system described above for maximum profit scheduling is 2-extendible for any  $L > 1$ .*

*Proof.* Let  $C + (i, t)$  be a feasible schedule and  $D$  an extension of  $C$ . Adding  $i$  at time  $t$  to  $D$  may create some conflicts, which can be fixed by removing the jobs  $i$  overlaps with. Since all jobs have the same length, job  $i$  overlaps with at most two other jobs.  $\square$

**Maximum asymmetric traveling salesman problem:** We are given a complete directed graph with non-negative weights and we must find a maximum weight tour that visits every city exactly once. The problem is NP-hard; the best known approximation factor for it is  $\frac{5}{8}$  [18].

The elements of our subset system are the directed edges of the complete graph; a set is independent if its edges form a collection of vertex disjoint paths or a cycle that visits every vertex exactly once.

**Theorem 6 ([15]).** *The subset system for maximum ATSP is 3-extendible.*

*Proof.* As usual let  $C + (x, y)$  be independent, and  $D$  be an extension of  $C$ . First remove from  $D$  the edges (if any) out of  $x$  and into  $y$ , these are clearly at most two and not in  $C$ . If we add  $(x, y)$  to  $D$  then every vertex has in-degree and out-degree at most one, but there may be a non-Hamiltonian cycle which uses  $(x, y)$ . There must be an edge in the cycle, not in  $C$ , that we can remove to break it. Therefore we need to remove at most three edges in total.  $\square$

**Matroid intersection:** This last theorem shows a nice relationship between matroids and  $k$ -extendible systems.

**Theorem 7.** *The intersection of  $k$  matroids is  $k$ -extendible*

*Proof.* Let  $(E, \mathcal{L}_i)$  for  $1 \leq i \leq k$  be our  $k$  matroids and let  $\mathcal{L} = \cap_i \mathcal{L}_i$ . We need to show that for every  $C \subseteq D \in \mathcal{L}$  and  $x \notin C$  such that  $C + x \in \mathcal{L}$  there exist  $Y \subseteq D \setminus C$  with at most  $k$  elements such that  $D \setminus Y + x \in \mathcal{L}$ .

Since the above sets are in  $\mathcal{L}$  they are also in  $\mathcal{L}_i$ . By Theorem 1 these individual matroids are 1-extendible, therefore we can find  $Y_i$  with at most one element such that  $D \setminus Y_i + x \in \mathcal{L}_i$ . Set  $Y = \cup_i Y_i$ , clearly  $|Y| \leq k$  and for all  $i$  we have  $D \setminus Y + x \in \mathcal{L}_i$ , which implies independence with respect to  $\mathcal{L}$ .  $\square$

### 3 A linear time $\frac{1}{2}$ -approximation for $b$ -matching

Because maximum weight  $b$ -matching can be solved exactly in  $O(\sum b(v) \min(m \log n, n^2))$  time [14], Greedy should be regarded as a tradeoff: we sacrifice optimality in order to get a much simpler algorithm which runs in  $O(m \log n)$  time. This tradeoff can be further improved to obtain a linear time  $\frac{1}{2}$ -approximation, our solution builds upon the work of Drake and Hougardy [11]. Let  $b = \max_{v \in V} b(v)$ , in this section we show:

**Theorem 8.** *There is a  $O(bm)$  time  $\frac{1}{2}$ -approximation algorithm for  $b$ -matching.*

The main procedure of our algorithm, LINEAR-MAIN, iteratively calls FIND-WALK, which greedily finds a heavy walk. Starting at some vertex  $u$  we take the heaviest edge  $(u, v)$  out of  $u$ , delete it from the graph, reduce  $b(u)$  by one, and repeat for  $v$ . If at some point the  $b(\cdot)$  value of a vertex becomes zero we delete all the remaining edges incident to it.

As we construct the walk we decrease the  $b(\cdot)$  value of the vertices in the walk. Except for the endpoints every node will have its  $b(\cdot)$  value decreased by 1 for every two edges in the walk incident to it. This means that  $M$ , the set of all walks, is not a valid solution as we can only guarantee that  $\deg_M(u) \leq 2b(u)$  for every vertex  $u$ .

Now consider choosing every other edge in a walk starting with the first edge. For any vertex the number of chosen edges incident to it is at most how much its  $b(\cdot)$  value was decreased while finding this walk. The same holds for the complement of this set, that is, picking every other edge starting with the second edge. We can therefore split  $M$  into two sets  $M_1$  and  $M_2$  by taking alternating edges of individual walks. These are valid solutions to our problem since for every vertex  $u$  we have  $\deg_{M_i}(u) \leq b(u)$ . Because  $M = M_1 \cup M_2$ , picking the one with maximum weight we are guaranteed a solution with weight at least  $\frac{w(M)}{2}$ . We now concentrate our effort in showing that  $w(M)$  is an upper bound on the cost of the optimal solution.

Let  $M_{OPT}$  be the optimal solution. We can imagine including an additional step in the FIND-WALK( $u$ ) function in which an edge  $e \in M_{OPT}$  is assigned to the heavy edge  $(u, v)$ : If  $(u, v) \in M_{OPT}$  then we assign it to itself, otherwise we pick any edge  $e \in M_{OPT}$  incident to  $u$ . In either case after  $e$  is assigned we remove it from  $M_{OPT}$ , so that it is not later assigned to a different edge.

It may be that some edges in  $M$  do not receive any edge from  $M_{OPT}$ , but can an edge in  $M_{OPT}$  be left unassigned? The following lemma answers this question and relates the cost of the two edges.

**Lemma 2.** *The modified FIND-WALK procedure assigns every edge  $e \in M_{OPT}$  to a unique edge  $(u, v) \in M$ , furthermore  $w(e) \leq w(u, v)$ .*

*Proof.* Suppose, for the sake of contradiction, that  $(x, y) \in M_{OPT}$  was not assigned. It is easy to see that if the  $b(\cdot)$  value of some vertex  $u$  becomes 0 then all edges in  $M_{OPT}$  incident to  $u$

<pre> LINEAR-MAIN(<math>G, w</math>) 1 <math>M \leftarrow \emptyset</math> 2 <b>while</b> <math>\exists u \in V</math> such that    <math>b(u) &gt; 0</math> and <math>\deg(u) &gt; 0</math> 3 <b>do</b> <math>M \leftarrow M + \text{FIND-WALK}(u)</math> 4 <b>split</b> <math>M</math> into <math>M_1</math> and <math>M_2</math> 5 <b>return</b> <math>\text{argmax}\{w(M_i)\}</math> </pre>	<pre> FIND-WALK(<math>u</math>) 1 <math>b(u) \leftarrow b(u) - 1</math> 2 <b>if</b> <math>\deg(u) = 0</math> 3   <b>then return</b> <math>\emptyset</math> 4 <b>let</b> <math>(u, v)</math> be the heaviest edge out of <math>u</math> 5 <b>remove</b> <math>(u, v)</math> from <math>G</math> 6 <b>if</b> <math>b(u) = 0</math> 7   <b>then</b> <b>remove</b> all edges incident to <math>u</math> 8 <b>return</b> <math>(u, v) + \text{FIND-WALK}(v)</math> </pre>
---	--

**Fig. 1.** A linear time  $\frac{1}{2}$  approximation for  $b$ -matching

must be assigned. Thus when the algorithm terminated  $b(x), b(y) > 0$  and  $\deg(x) = \deg(y) = 0$ . Therefore the edge  $(x, y)$  must have been deleted from the graph because it was traversed (chosen in  $M$ ). In this case we should have assigned  $(x, y)$  to itself. We reached a contradiction, therefore all edges in  $M_{OPT}$  are assigned a unique edge in  $M$ .

If  $(x, y)$  was assigned to itself then the lemma follows, suppose then that it got assigned to  $(x, v)$  in the call FIND-WALK( $x$ ). Notice that at the moment the call was made  $b(x), b(y) > 0$ . If at this moment  $(x, y)$  was present in the graph the lemma follows as  $(x, v)$  is the heaviest edge out of  $x$ . We claim this is the only alternative. If  $(x, y)$  had been deleted before it would be because it was traversed and thus it should have been assigned to itself.  $\square$

An immediate corollary of Lemma 2 is that  $w(M_{OPT}) \leq w(M)$ , which as mentioned implies the algorithm returns a solution with cost at least  $\frac{w(M_{OPT})}{2}$ . Now we turn our attention to the time complexity.

The running time is dominated by the time spent finding heavy edges. This is done by scanning the adjacency list of the appropriate vertex. An edge  $(x, y)$  may be considered several times while looking for a heavy edge out of  $x$  and  $y$ . The key observation is that this can happen at most  $b(x) + b(y)$  times. Each time we reduce the value of either endpoint by one, when one of them reaches 0 all edges incident to that endpoint are deleted and after that  $(x, y)$  is never considered again. Adding up over all edges we get a total time of  $O(bm)$ .

## 4 A randomized $(\frac{2}{3} - \epsilon)$ -factor algorithm

In this section we generalize ideas from Pettie and Sander [21] to improve the approximation ratio of our linear time algorithm. We will develop a randomized algorithm that returns a solution with expected weight at least  $(\frac{2}{3} - \epsilon) w(M_{OPT})$  and runs in expected  $O(bm \log \frac{1}{\epsilon})$  time.

Before describing the algorithm we need to define a few terms, all of which are with respect to a given solution  $M$ . An edge  $e$  is *matched* if  $e \in M$  otherwise we say  $e$  is *free*. A set of edges  $S$  can be used to update the matching by taking the symmetric difference of  $M$  and  $S$  denoted by  $M \oplus S = (M \cup S) \setminus (M \cap S)$ . The set  $S$  is said to be *compatible* with  $M$  if  $M \oplus S$  is a valid  $b$ -matching.

Our algorithm works by iteratively finding a compatible set of edges and updating our current solution  $M$  with it. To keep the running time low we only look for *arms* and *pieces*. An *arm*  $A$  out of a vertex  $u$  consists of a free edge  $(u, x)$  followed, maybe, by a matched edge  $(x, y)$ . The *benefit* of  $A$  is defined as  $w(u, x) - w(x, y)$ , note that  $\text{benefit}(A) = w(M \oplus A) - w(M)$ . Let  $(u, v) \in M$ , a *piece*  $P$  about  $(u, v)$  consists of the edge  $(u, v)$ , and, possibly, of arms  $A_u$  and  $A_v$  out of  $u$  and  $v$ . The benefit of the piece is defined as  $\text{benefit}(A_u) + \text{benefit}(A_v) - w(u, v)$ . Notice that if  $A_u$  and  $A_v$  use the same matched edge then  $\text{benefit}(P) < w(M \oplus P) - w(M)$ , otherwise these two quantities are the same.

We now describe in detail an iteration of our algorithm. First we pick a vertex  $u$  uniformly at random. Then we probabilistically decide to either: choose an edge  $(u, v) \in M$  and augment  $M$  using a max-benefit compatible piece about  $(u, v)$ , augment  $M$  with a max-benefit compatible arm out of  $u$ , or simply do nothing. See Fig. 2 for the exact probabilities of these events. This is repeated  $k$  times, the parameter  $k$  will be determined later to obtain:

**Theorem 9.** *The procedure LINEAR-RANDOM finds a  $b$ -matching in  $O(bm \log \frac{1}{\epsilon})$  time with expected weight at least  $(\frac{2}{3} - \epsilon) w(M_{OPT})$ .*

Let us first prove the approximation ratio of LINEAR-RANDOM. Our plan is to construct a set  $Q$  of pieces and arms with benefit at least  $2w(M_{OPT}) - 3w(M)$  and then argue that the expected gain of each iteration is a good fraction of this. Note that if  $2w(M_{OPT}) - 3w(M) \leq 0$  then  $M$  is already a  $\frac{2}{3}$ -approximate solution. In what follows we assume without loss of generality that  $M_{OPT}$  and  $M$  are disjoint—any overlap only makes our bounds stronger.

```

LINEAR-RANDOM( $G, w$ )
1  $M \leftarrow \emptyset$ 
2 do
3   pick a vertex  $u$  uniformly at random
4   with prob  $\frac{\deg_M(u)}{b}$  do
5     pick  $(u, v) \in M$  uniformly at random
6     find max-benefit compatible piece  $P$  about  $(u, v)$ 
7      $M \leftarrow M \oplus P$ 
8   with prob  $\frac{b(u) - \deg_M(u)}{b}$  do
9     find max-benefit compatible arm  $A$  out of  $u$ 
10     $M \leftarrow M \oplus A$ 
11 repeat  $k$  times

```

**Fig. 2.** A linear time  $(\frac{2}{3} - \epsilon)$ -factor algorithm for  $b$ -matching

In order to construct  $Q$  we need to pair edges of  $M_{OPT}$  and  $M$ . Every edge  $(u, v) \in M_{OPT}$  is paired with  $(u, x) \in M$  via  $u$  and  $(v, y) \in M$  via  $v$  in such a way that every edge in  $M$  is paired with at most two edges, one via each endpoint. If  $\deg_{M_{OPT}}(u) > \deg_M(u)$  then the excess of  $M_{OPT}$  edges are assigned to  $u$ . Thus every edge  $(u, v) \in M_{OPT}$  is paired/assigned exactly twice, once via each endpoint.

For every edge  $(u, x) \in M$  we build a piece  $P$  by finding arms  $A_u$  and  $A_x$  out of  $u$  and  $x$ . To construct  $A_u$  follow, if any, the edge  $(u, y) \in M_{OPT}$  paired with  $(u, x)$  via  $u$ , then take, if any, the edge  $(y, z) \in M$  paired with  $(u, y)$  via  $y$ . A similar procedure is used to construct  $A_x$ . Finally we assign  $P$  to vertex  $u$  and add it to  $Q$ . Also for every  $u \in V$  which has been assigned edges  $(u, v) \in M_{OPT}$  we grow an arm  $A$  out of  $u$  using  $(u, v)$ . These arms are assigned to  $u$  and added to  $Q$ .

Every edge in  $M_{OPT}$  appears in exactly two of the pieces and arms in  $Q$ , on the other hand every edge in  $M$  appears at most three times. Therefore the benefit of  $Q$  is at least  $2w(M_{OPT}) - 3w(M)$ .

How many pieces/arms can be assigned to a single vertex  $u$ ? At most  $\deg_M(u)$  pieces, one per  $(u, x) \in M$ , and at most  $b(u) - \deg_M(u)$  arms, one per  $(u, v) \in M_{OPT}$  which did not get paired up with  $M$  edges via  $u$ . A simple case analysis shows that all these pieces and arms are compatible with  $M$ . Therefore the expected benefit of the piece or arm picked in any given iteration is:

$$\begin{aligned}
\mathbb{E}[\text{benefit}] &= \frac{1}{n} \sum_{u \in V} \frac{b(u) - \deg_M(u)}{b} \text{max-arm}(u) + \sum_{(u,v) \in M} \frac{1}{b} \text{max-piece}(u, v) \\
&\geq \frac{1}{bn} \sum_{u \in V} \text{benefit of pieces/arms assigned to } u \\
&\geq \frac{1}{bn} \text{benefit}(Q) \\
&\geq \frac{3}{bn} \left( \frac{2}{3} w(M_{OPT}) - w(M) \right)
\end{aligned}$$

From this inequality we can derive the following lemma which is very similar to Lemma 3.3 from [21], we include its proof for completeness.

**Lemma 3.** *After running LINEAR-RANDOM for  $k$  iterations  $M$  has an expected weight of at least  $\frac{2}{3}w(M_{OPT})(1 - e^{-\frac{3k}{bn}})$*

*Proof.* Let  $X_i = \frac{2}{3}w(M_{OPT}) - w(M_i)$ , where  $M_i$  is the matching we get at the end of the  $i$ th iteration. From the above inequality and the fact that the gain of each iteration is at least as much as the benefit of the piece/arm found we can infer that  $E[X_{i+1}|X_i] \leq X_i - \frac{3}{bn}X_i$ . Thus  $E[X_{i+1}] \leq E[X_i] \left(1 - \frac{3}{bn}\right)$ , and

$$E[X_k] \leq E[X_0] \left(1 - \frac{3}{bn}\right)^k \leq \frac{2}{3}w(M_{OPT}) e^{-\frac{3k}{bn}}.$$

By setting  $k = \frac{bn}{3} \log \frac{1}{\epsilon}$  we get a matching with expected cost at least  $\left(\frac{2}{3} - \epsilon\right) w(M_{OPT})$ . Let us now turn our attention to the running time.

To compute a max benefit arm out of a vertex  $u$  we follow free edges  $(u, v)$  and if  $\deg_M(v) = b(v)$  we scan the list of matched edges incident to  $v$  to find the lightest such edge; among the arms found we return the best. Notice that this can take as much as  $O(b \deg(v))$  time. Suppose now, that we already had computed for every vertex which is the lightest matched edge incident to it, then the task can be carried out in just  $O(\deg(v))$  time.

To produce a max benefit piece about  $(u, v)$  we can try finding max benefit arms out of  $u$  and  $v$  in  $O(\deg(u) + \deg(v))$  time. This unfortunately does not always work as the resulting piece may not be compatible, consider finding arms  $\{(u, x)\}$  and  $\{(v, x)\}$  with  $\deg_M(x) = b(x) - 1$ , or  $\{(u, x), (x, z)\}$  and  $\{(v, x), (x, z)\}$  with  $\deg_M(x) = b(x)$ ; both arms are compatible by themselves, but  $x$  cannot take both at once. If this problem arises, it can be solved by taking the best arm for  $u$  and the second best arm for  $v$ , or the other way around, and keeping the best pair. To find the second best arm we need to have access to the second lightest matched edge incident to any vertex.

Once we found our piece/arm we have to update the matching. This may change the lightest matched edges incident to vertices on the piece/arm. Since there are most 6 such vertices the update can be carried out in  $O(b)$  time. The expected work done in a single iteration is given by:

$$\begin{aligned} E[\text{work}] &\leq \frac{1}{n} \sum_{u \in V} \frac{b(u) - \deg_M(u)}{b} (\deg(u) + b) + \sum_{(u,v) \in M} \frac{1}{b} (\deg(u) + \deg(v) + b) \\ &\leq \frac{1}{n} \sum_{u \in V} \deg(u) + b(u) + \sum_{(u,v) \in M} \frac{1}{b} \deg(v) \leq \frac{3}{n} \sum_{u \in V} \deg(u) = \frac{6m}{n} \end{aligned}$$

The third inequality assumes  $b(u) \leq \deg(u)$ . If this is not the case we can just set  $b(u)$  to be  $\deg(u)$  which does not change the optimal solution.

There are  $k = \frac{bn}{3} \log \frac{1}{\epsilon}$  iterations each taking  $O(\frac{m}{n})$  time, by linearity of expectation the total expected running time is  $O(bm \log \frac{1}{\epsilon})$ .

## 5 Conclusion

We introduced the notion of  $k$ -extendible systems which allowed us to explain the performance of the greedy algorithm on seemingly disconnected problems. We also provided better approximation algorithms for  $b$ -matching, a specific problem that falls in our framework. It would be interesting to improve the approximation factor of other problems in this class beyond  $\frac{1}{k}$ .

**Acknowledgments:** Thanks to Hal Gabow and Allan Borodin for their encouraging words and for providing references to recent work on approximating maximum weight matching and priority algorithms. Special thanks to Samir Khuller for pointing out the problem and providing comments on earlier drafts.

## References

1. S. Angelopoulos and A. Borodin. The power of priority algorithms for facility location and set cover. *Algorithmica*, 40(4):271–291, 2004.
2. E. M. Arkin and R. Hassin. On local search for weighted k-set packing. *Mathematics of Operations Research*, 23(3):640–648, 1998.
3. V. Arora, S. Vempala, H. Saran, and V. V. Vazirani. A limited-backtrack greedy schema for approximation algorithms. In *FSTTCS*, pages 318–329, 1994.
4. P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal. *Journal of Scheduling*, 2:245–252, 1999.
5. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
6. A. Borodin, M. N. Nielsen, and C. Rockoff. (Incremental) Priority algorithms. *Algorithmica*, 37(4):295–326, 2003.
7. A. V. Borovik, I. Gelfand, and N. White. Symplectic matroid. *Journal of Algebraic Combinatorics*, 8:235–252, 1998.
8. A. Bouchet. Greedy algorithm and symmetric matroids. *Mathematical Programming*, 38:147–159, 1987.
9. M. Chrobak, C. Dürr, W. Jawor, L. Kowalik, and M. Kurowski. A note on scheduling equal-length jobs to maximize throughput. *Journal of Scheduling*, 9(1):71–73, 2006.
10. D. E. Drake and S. Hougardy. Improved linear time approximation algorithms for weighted matchings. In *APPROX*, pages 14–23, 2003.
11. D. E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85:211–213, 2003.
12. J. Edmonds. Minimum partition of a matroid into independent subsets. *J. of Research National Bureau of Standards*, 69B:67–77, 1965.
13. J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–36, 1971.
14. H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983.
15. T. A. Jenkyns. The greedy travelling salesman’s problem”. *Networks*, 9:363–373, 1979.
16. B. Korte and D. Hausmann. An analysis of the greedy algorithm for independence systems. *Ann. Disc. Math.*, 2:65–74, 1978.
17. B. Korte and L. Lovász. Greedoids—a structural framework for the greedy algorithm. In *Progress in Combinatorial Optimization*, pages 221–243, 1984.
18. M. Lewenstein and M. Sviridenko. Approximating asymmetric maximum TSP. In *SODA*, pages 646–654, 2003.
19. L. Lovász. The matroid matching problem. In *Algebraic Methods in Graph Theory*, Colloquia Mathematica Societatis Janos Bolyai, 1978.
20. J. G. Oxley. *Matroid Theory*. Oxford University Press, 1992.
21. S. Pettie and P. Sanders. A simpler linear time  $2/3 - \epsilon$  approximation to maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.
22. R. Preis. Linear time  $1/2$ -approximation algorithm for maximum weighted matching in general graphs. In *STACS*, pages 259–269, 1999.
23. R. Rado. A theorem on independence relations. *Quart. J. Math.*, 13:83–89, 1942.
24. A. Schrijver. *Combinatorial Optimization*. Springer, 2003.
25. A. Vince. A framework for the greedy algorithm. *Discrete Applied Mathematics*, 121(1-3):247–260, 2002.
26. H. Whitney. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57:509–533, 1935.