Title: Matroids: The Theory and Practice of Greed

Authors:

Christian Jones
Department of Mathematics
University of Florida
Gainesville, FL 32611
chjones@ufl.edu

Ran Libeskind-Hadas
Department of Computer Science
Harvey Mudd College
Claremont, CA 91711
hadas@cs.hmc.edu

Mathematical Field: Linear algebra, graph theory, computer science

Application Field: Computer science, discrete optimization

Target Audience: Students in a course in linear algebra, discrete mathematics, graph theory, or algorithms.

Abstract: A matroid is a mathematical structure that generalizes the notion of linear independence. Remarkably, this simple and elegant mathematical structure can be used to systematically develop efficient and simple "greedy" algorithms for a variety of discrete optimization problems. Moreover, matroids provide some insight into why other discrete optimization problems are apparently computationally intractable. This module introduces matroids and demonstrates their application to several discrete optimization problems.

Prerequisites: The reader is assumed to be familiar with elementary concepts in linear algebra (definition and properties of linear independence) and elementary concepts in graph theory (definition of a graph, bipartite graph, and path).

Matroids: The Theory and Practice of Greed

Christian Jones Ran Libeskind-Hadas

Introduction

This paper shows how some natural generalizations of concepts from linear algebra can be used to find simple and efficient algorithms for many discrete optimization problems. Specifically, we describe a mathematical structure called a *matroid*. We then show how matroids can be used to construct so-called *greedy* algorithms for a variety of discrete optimization problems.

We begin by considering the case of a new long-distance telephone company. The company plans to offer service between n cities but to lease the actual phone lines from existing companies. A direct phone line exists between certain pairs of cities and there is a positive cost associated with leasing each line. The new company would like to lease a subset of lines such that it can provide a path between any two cities using only the leased lines. A *solution* is any subset of lines with this property. An *optimal solution* is a solution of minimum total cost. For example, consider the weighted graph in Figure 1(a) where vertices correspond to cities, edges correspond to existing phone lines, and the edge weights correspond to the leasing costs of the phone lines. One possible solution is shown in Figure 1(b) with a total cost of 18. Another solution is shown in Figure 1(c) with a total cost of only 11. It is possible to verify that the solution in Figure 1(c) is an optimal solution.

Note that any solution must span all of the vertices in the graph and an optimal solution must be a tree: a connected graph with no cycles. If a solution contains a cycle then any edge on the cycle can be removed without destroying connectivity while decreasing the cost of the solution. Thus, given a connected graph, an optimal solution is a spanning tree of minimum total cost, also known as a *minimum spanning tree*.

The minimum spanning tree problem was studied as early as 1926 by the Czech mathematician Otakar Borůvka in connection with minimizing the cost of electric networks (Borůvka, 1926). The problem was later studied by Prim (Prim, 1957) and Kruskal (Kruskal, 1956) among others. Kruskal showed that the following simple algorithm can be used for finding a minimum spanning tree in a connected graph: Let S be an initially empty set. Sort the edges of the graph in order of nondecreasing edge costs. Consider each edge in the sorted list, beginning with an edge of least cost, and add the edge to S if and only if it does not create a cycle with the edges already in S. Kruskal showed that after all edges have been considered, S is a minimum spanning tree for the graph. For example, in the graph in Figure 1, Kruskal's algorithm sorts the edges in the order $1, 2, 3, 4, 5, 5, 6$. It begins by selecting the edge of weight 1 and adding it to S. The edge of weight 2 is added next, followed by the edge of weight 3. The edge of weight 4 is considered next, but cannot be

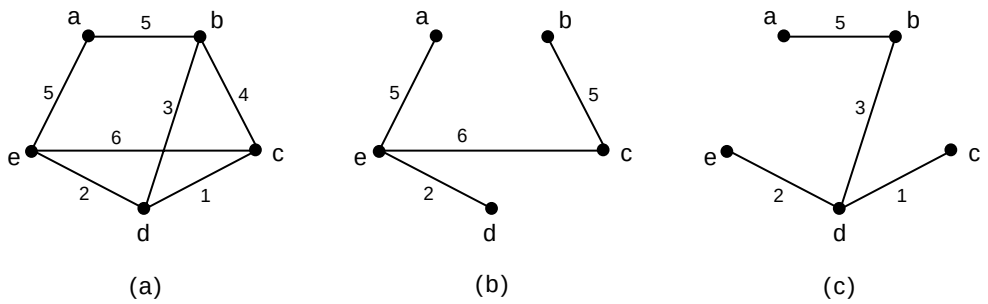

Figure 1: (a) A weighted graph with five vertices. (b) A solution with cost 18. (c) An optimal solution with cost 11.

added to S because it creates a cycle with existing edges in S. Next, one of the two edges of weight 5 will be considered and added to S. The next edge of weight 5 cannot be added to S because it creates a cycle with existing edges. Similarly, the edge of weight 6 cannot be added. At this point the algorithm terminates and S is a minimum spanning tree.

Kruskal's algorithm is said to be "greedy" because at each step it simply chooses the cheapest remaining edge that does not introduce a cycle. In general, a "greedy" algorithm is one that makes a sequence of locally optimal decisions. In the case of the minimum spanning tree problem, this sequence of locally optimal decisions leads to a globally optimal solution. Unfortunately, for other problems, greedy algorithms do not always find optimal solutions.

Consider the famous and seemingly similar *traveling salesperson problem*. In this problem we are given a graph with n vertices representing n cities. There is an edge between every pair of vertices with an associated positive real number representing the cost of a direct flight between the corresponding cities. A salesperson wishes to start at her home city, visit each city at least once, and return to her home city. However, since she is very busy, the salesperson stipulates that she will not fly to any city more than once. Thus, the salesperson wishes to find a cycle that visits each city exactly once: a *traveling salesperson tour* or *Hamiltonian cycle* in the graph. Among all tours, the salesperson wants to find one which minimizes the sum of the edge costs. For example, consider the weighted graph with five vertices, representing five cities, in Figure 2(a). One tour is shown in Figure 2(b) and has a total cost of 12. Another tour is shown in Figure 2(c) and has a cost of 7. It can be shown that the tour in Figure 2(c) is an optimal tour.

It seems intuitively natural to try to solve the traveling salesperson problem using a greedy algorithm. For example, starting at the home city, we might "greedily" select the least expensive flight leaving that city. Assume that this flight brings us to city v. We could now "greedily" select the cheapest flight from city v that brings us to a city that we have not yet visited. The process can be repeated until we reach a city such that all other cities have been visited. At this point we are forced to fly directly to the starting city and the tour is complete.

Surprisingly, this greedy approach for the traveling salesperson problem does not always find optimal solutions. For example, consider the graph with four vertices in Figure 3(a). If vertex a represents the start city, the greedy algorithm selects vertex b as the next city

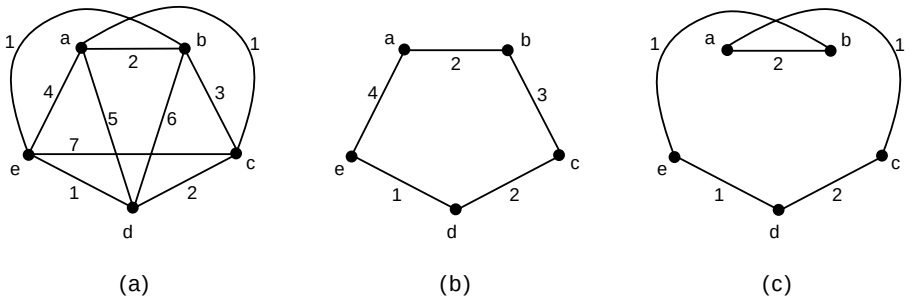

Figure 2: (a) A weighted graph with five vertices. (b) A tour with cost 12. (c) An optimal tour with cost 7.

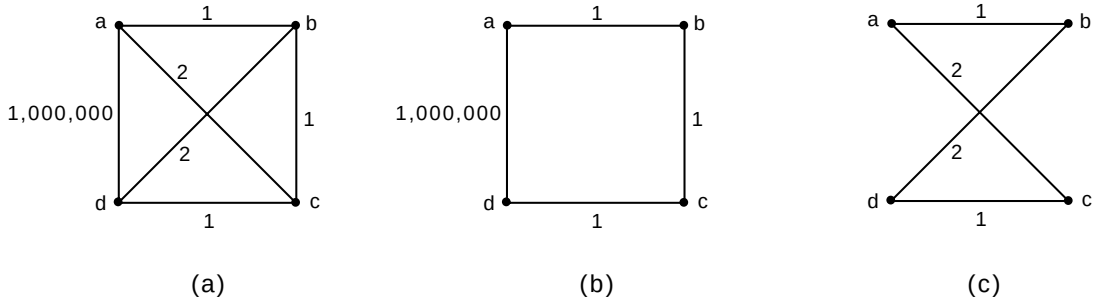

Figure 3: (a) A weighted graph with four vertices. (b) A solution with cost $1,000,003$. (c) An optimal solution with cost 6.

followed by vertex c, then vertex d, and finally returing to vertex a. The cost of this tour, shown in Figure 3(b) is $1,000,003$. On the other hand, the tour in Figure 3(c) has a cost of 6.

Not only does this particular greedy algorithm fail for the traveling salesperson problem, but no efficient algorithm is known for this problem.[1] In fact, the traveling salesperson problem is a *NP-complete* problem. This means, roughly, that not only is no efficient algorithm known for the problem but that the discovery of an efficient algorithm would immediately imply efficient algorithms for a multitude of other apparently intractable computational problems.

While many discrete optimization problems can be solved by greedy algorithms, many others seem not to be amenable to greed or even any efficient algorithm at all. This article describes how an elegant mathematical structure, called a *matroid*, can be used to construct and establish the correctness of greedy algorithms for a variety of discrete optimization problems. Moreover, matroids also help us understand why certain other problems are not amenable to efficient solution.

In the following sections of this paper, we begin by defining the concept of a matroid.

[1] An algorithm is said to be *efficient* if its running time is polynomial in the size of the problem instance. The size of the problem instance is the number of digits required to encode it in binary. See (Garey & Johnson, 1979) or (Cormen *et al.*, 1990) for more on this topic.

Next, we show how matroids are directly related to greedy algorithms. We then use matroids to find greedy algorithms for several optimization problems. We conclude with a discussion of the relationship between matroids and intractable problems such as the traveling salesperson problem and give a brief overview of some other mathematical structures related to matroids and their applications.

Matroids

A matroid is a mathematical structure, introduced by Whitney in 1935 (Whitney, 1935), that generalizes the notion of linear independence. Recall that in any vector space, an independent set of vectors has the property that each of its subsets is also an independent set. In addition, if X and Y are two independent sets such that $|X| > |Y|$ then there exists some element $e \in X - Y$ such that $Y + e$ is also an independent set.[2]

Remarkably, there are many sets, other than vector spaces, with their own associated definitions of "independence" that satisfy the two properties above. A matroid is any structure that satisfies these properties.

Definition 1 *A* matroid *is an ordered pair $M = (E, \mathcal{I})$ where E is a finite set and I is a set of subsets of E satisfying the following two properties:*

Heredity Property: *The empty set is in $\mathcal{I}$ and for any set $X \in \mathcal{I}$ all subsets of X are also elements of $\mathcal{I}$.*

Exchange Property: *If $X, Y \in \mathcal{I}$ such that $|X| > |Y|$ then there exists some $e \in X - Y$ such that $Y + e \in \mathcal{I}$.*

The elements of set $\mathcal{I}$ are called, not surprisingly, the *independent sets* of M.

As an example of a matroid, consider any matrix whose elements are real numbers. Let E be the set of rows of the matrix and let $\mathcal{I}$ be the set of all linearly independent subsets of E. Now, $M = (E, \mathcal{I})$ is easily verified to be a matroid. In fact, the name "matroid" comes from this relationship with matrices.

A more interesting example of a matroid is one induced from a graph, known as the *graphic matroid*. Consider a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. If $E' \subseteq E$ and $F = (V, E')$ contains no cycles, then F is said to be a *spanning forest* of G. Each connected component in the spanning forest is called a *tree*. For any graph $G = (V, E)$, let $\mathcal{I}$ be the set of all $E' \subseteq E$ such that (V, E') is a spanning forest of G. We claim that $M_G = (E, \mathcal{I})$ is a matroid.

The heredity property of matroids is easily seen to hold for M_G: Since $(V, \emptyset)$ is a spanning forest of G, $\emptyset \in \mathcal{I}$. In addition, if $E' \in \mathcal{I}$ then (V, E') is a spanning forest and thus (V, E'') is a spanning forest for any $E'' \subseteq E'$.

The exchange property requires slightly more work to verify. Assume that graph G has n vertices and let $F = (V, E')$ be a spanning forest in G. Note that if $E' = \emptyset$ then it contains no

[2]Throughout this paper, the notation $|A|$ denotes the cardinality of set A, $A - B$ denotes the set of elements in set A that are not in set B, and $A + e$ denotes the set formed by adding element e to set A.

edges and thus comprises n distinct trees, each of which is a single vertex. If $|E'| = 1$, then F comprises $n - 1$ trees: One tree is two vertices connected by an edge and the remaining $n - 2$ trees are distinct vertices. In general, if an edge $e \notin E'$ does not create a cycle when added to E', then the edge must connect two distinct trees in F. Therefore $F + e$ comprises one fewer tree than does F. Thus, in general, a forest with $k < n$ edges comprises exactly $n - k$ trees.

In relation to the graphic matroid, let X and Y be two edge sets in $\mathcal{I}$ such that $|X| > |Y|$. Thus, $F_X = (V, X)$ and $F_Y = (V, Y)$ are both spanning forests of G. By the above observation, F_X comprises fewer trees than does F_Y. Therefore, there is some tree in F_X whose vertices are in more than one tree of F_Y. This means that there is an edge $e \in X$ whose endpoints are in different trees of F_Y, and therefore $F_Y + e$ does not contain a cycle and is therefore a spanning forest. Thus, M_G is indeed a matroid. For example, for the graph in Figure 1 some of the independent sets are $\{ab\}$, $\{ab, bc, de\}$, and $\{ae, ce, de, bc\}$, and $\{ab, bd, cd, de\}$

Before turning to applications of matroids to discrete optimization problems, we note that the analogy between matroids and vector spaces does not end with independent sets. A related analogy, which will be used extensively in the next section, is that of a *basis*. In a vector space, a basis is a maximal independent set; an independent set such that the addition of any other vector to this set results in a set that is no longer independent. Similarly, a *basis* in a matroid $M = (E, \mathcal{I})$ is defined to be a maximal independent set; an element $I \in \mathcal{I}$ such that $I + e \notin \mathcal{I}$ for all $e \in E - I$. A fundamental result of vector spaces is that all bases have the same size. Analogously, we can directly apply the definition of a matroid to prove the following lemma.

Lemma 1 *All bases of a matroid have the same size.*

The proof of this lemma and other analogies between vector spaces and matroids are explored in the exercises.

From Matroids to Greedy Algorithms

We are now ready to establish the connection between matroids and greedy algorithms. In a discrete optimization problem, each element typically has some associated cost or *weight* and we wish to find a solution of minimum or maximum total weight. In the minimum spanning tree problem, for example, a weight is associated with each edge in the graph and we wish to find a spanning tree of minimum total weight.

For a given matroid $M = (E, \mathcal{I})$, let w be a weight function that assigns a real number $w(x)$ to each $x \in E$. We can easily extend the definition of weight to apply to sets of elements: For any set $S \subseteq E$, we define the weight of the set S to be

$$w(S) = \sum_{x \in S} w(x).$$

To see how this extension of the weight function is useful, we revisit the the minimum spanning tree problem. For a given instance of the minimum spanning tree problem we can

construct a corresponding graphic matroid $M_G = (E, \mathcal{I})$. Recall that E is the set of edges in the graph and $\mathcal{I}$ is the set of all spanning forests in the graph. Let w be a weight function that assigns a positive weight to each edge in the graph. Then we claim that the objective of the minimum spanning tree problem is that of finding a basis of M_G such that $w(X)$ is minimized. To see this, recall that a basis is a maximal independent set; an element in $\mathcal{I}$ such that no proper superset of this element is in $\mathcal{I}$. In this case, elements in $\mathcal{I}$ are spanning forests and thus a maximal element is a forest such that no edge can be added without creating a cycle. Such a forest is a spanning tree. Thus, a basis of minimum weight in the graphic matroid is a minimum spanning tree in the graph.

A remarkable property of matroids is that a basis of maximum weight can be found using a simple greedy algorithm. (The case of finding a basis of minimum weight will be shown to follow easily from this.) Given a matroid $M = (E, \mathcal{I})$ and a weight function $w : E \rightarrow \mathbb{R}$ the *matroid greedy algorithm* performs the following steps:

Sort the n elements of E into list $e_1, e_2, \ldots, e_n$ such that $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_n)$
Let $X = \emptyset$
for $i = 1$ **to** n
  if $X + e_i \in \mathcal{I}$
    then let $X = X + e_i$
return X

Theorem 1 *For any matroid $M = (E, \mathcal{I})$ and weight function $w : E \rightarrow \mathbb{R}$ the matroid greedy algorithm returns a basis of maximum weight.*

Proof: First, we observe that the set X returned by the algorithm is a basis of M. If not then there exists $e_i \in E - X$ such that $X + e_i \in \mathcal{I}$. By the heredity property, every subset of $X + e_i$ is in $\mathcal{I}$ and thus e_i would have been added to the set X in step i of the **for** loop, a contradiction.

Let $X = \{x_1, \ldots, x_k\}$ where $w(x_1) \geq w(x_2) \geq \ldots \geq w(x_k)$. Let Y be a basis of M of maximum weight. By Lemma 1, $|Y| = k$. Let $Y = \{y_1, \ldots, y_k\}$ where $w(y_1) \geq w(y_2) \geq \ldots \geq w(y_k)$. If $w(x_i) \geq w(y_i)$ for all i, $1 \leq i \leq k$, then X is also a basis of maximum weight. Assume therefore that this is not the case and let ℓ be the least value such that $w(x_\ell) < w(y_\ell)$. Consider the sets

$$X_{\ell-1} = \{x_1, \ldots, x_{\ell-1}\} \text{ and } Y_\ell = \{y_1, \ldots, y_\ell\}.$$

By the heredity property these sets are independent. The exchange property implies that there exists some $y_i \in Y_\ell - X_{\ell-1}$ such that $X_{\ell-1} + y_i \in \mathcal{I}$. Since $w(y_i) \geq w(y_\ell) > w(x_\ell)$, the greedy algorithm considers y_i before x_ℓ. By the heredity property, every subset of $X_{\ell-1} + y_i$ is in $\mathcal{I}$ and thus the algorithm would have included y_i in X, a contradiction. □

As an example, we now revisit the minimum spanning tree problem. As discussed earlier, the minimum spanning tree problem is exactly that of finding a basis of minimum weight in the graphic matroid $M_G = (E, \mathcal{I})$. Assume for a moment that we actually wanted to find a spanning tree of maximum weight; a *maximum spanning tree*. In this case, Theorem 1 tells us that we can simply apply the matroid greedy algorithm to the corresponding graphic matroid. In other words, we begin by sorting the edges in E in order of nonincreasing

weights. Beginning with an initially empty set X, we consider the sorted edges in E one at a time. If an edge under consideration does not create a cycle with respect to the edges already in X, we add the edge to X. This results in a basis of maximum weight which is a maximum spanning tree. Now observe that a minimum spanning tree can be found by replacing each $w(e)$ in the graph by its negative, $-w(e)$, and finding a maximum spanning tree in this reweighted graph. Note that this is exactly equivalent to the aforementioned algorithm due to Kruskal.

In the next two sections we explore two more discrete optimization problems. For each of these problems, we find a corresponding matroid. We can then apply the matroid greedy algorithm to solve each of these problems.

A Scheduling Problem

Consider a set $S = \{1, \ldots, n\}$ of n jobs that must be performed by a single machine. Each job takes one unit of time to complete and once a job is started on the machine it must be completed before the next job can be started. For each job there is a deadline $d(i)$ such that $1 \leq d(i) \leq n$, $1 \leq i \leq n$. For each job there is also an associated positive real reward or weight, $w(i)$, that is obtained if the job is completed no later than its deadline. The objective is to determine a *schedule*, an ordering of the n jobs, that maximizes the total reward. In practice, these jobs might be mechanical tasks performed on a machine or programs run on a computer and the rewards might represent profits earned by completing the jobs by their deadlines. We solve this problem by constructing a corresponding matroid, showing that an optimal solution to the scheduling problem corresponds to a basis of maximum weight in the matroid, and then finding such a basis using the matroid greedy algorithm.

Given a schedule for the n jobs, we say that a job is *on time* if it is completed on or before its deadline and otherwise the job is said to be *late*. Let S be a set of jobs with associated deadlines and weights. A *feasible schedule* for $X \subseteq S$ is a schedule in which all of the jobs in X are on time. A subset X of S is said to be *feasible* if there exists at least one feasible schedule for X. Notice that the problem of finding an optimal schedule for S can be reduced to that of finding a feasible subset $X \subseteq S$ of maximum total weight. We would then like to find a feasible schedule for X. The elements in $S - X$ will all be late and can therefore be scheduled in any order after the jobs in X. Of course, we will need some way of determining whether a subset $X \subseteq S$ is feasible and, if it is, we will need to find a feasible schedule for it.

Imagine for a moment that we are given a set $X \subseteq S$ and told that X is feasible. We know that a feasible schedule exists for X, but it would seem that we might need to test all of the permutations of the elements of X, one by one, until we find a feasible schedule. This, of course, would be a prohibitively slow process. Surprisingly, if X is known to be feasible, it is very easy to find a feasible schedule: we simply sort the jobs in order of nondecreasing deadlines. Perhaps even more remarkable is the fact that to determine whether or not a set X is feasible in the first place, we need only sort the jobs in order of nondecreasing deadlines and check to see if every job is on time in this schedule. We now formalize these claims in the following lemma.

Lemma 2 *A set X is feasible if and only if the schedule formed by sorting the elements of*

X in order of nondecreasing deadlines results in each job being on time.

Proof: Assume that X is feasible. Then there exists some schedule that completes the jobs in X on time. If this schedule has some pair of jobs i and j both of which are completed on time but such that i is completed before j and $d(i) > d(j)$, then we swap the two jobs. Now, job j is completed even earlier and is therefore still on time. Moreover, job i is now completed when j was completed in the original schedule and is therefore completed no later than time $d(j)$. Since $d(i) > d(j)$, job i is still on time in this new schedule. Therefore, given any schedule that completes all of the jobs in X on time, we can repeatedly swap pairs of on-time jobs until they are completed in order of nondecreasing deadlines with all jobs still being completed on time.

Conversely, assume that for a given set X, the schedule formed by sorting the elements of X in order of nondecreasing deadlines results in each job being on time. Since this is a feasible schedule for X, X is feasible by definition. □

Let us define $\mathcal{I}$ to be the set of all feasible subsets of S. We will next show that $M_S = (S, \mathcal{I})$ is a matroid. Since all weights are positive in this problem, a feasible set of maximum weight is precisely a basis of maximum weight in the matroid. Thus, the matroid greedy algorithm can be applied to solve this scheduling problem.

Lemma 3 *Given a set of jobs S, let $\mathcal{I}$ denote the set of all feasible subsets of S. Then $M_S = (S, \mathcal{I})$ is a matroid.*

Proof: The heredity property is satisfied because $\emptyset$ is trivially feasible and every subset of a feasible set is clearly also feasible.

To show that the exchange property is satisfied, let X and Y be two elements of $\mathcal{I}$ such that $|X| > |Y|$. Without loss of generality, assume that $|X| = |Y| + 1$. (If this is not the case, we simply remove elements from X arbitrarily until this assumption is true.) Let $|Y| = n$ and let $x_1, \ldots, x_{n+1}$ and $y_1, \ldots, y_n$ denote the elements of X and Y, respectively, in order of nondecreasing deadlines. That is, $d(x_i) \leq d(x_{i+1})$, $1 \leq i \leq n$, and $d(y_j) \leq d(y_{j+1})$, $1 \leq j \leq n-1$. If $x_{n+1} \notin Y$ then $x_{n+1} \in X - Y$ and the schedule $y_1, \ldots, y_n, x_{n+1}$ is a feasible schedule for $Y + x_{n+1}$ because x_{n+1} completes at time $n+1$ in the schedule for X and thus $d(x_{n+1}) \geq n+1$.

Assume, therefore, that $x_{n+1} \in Y$. Since $|X| > |Y|$, there exists some element of X that is not in Y. Let k be the largest value of i such that $x_i \notin Y$. Then $1 \leq k \leq n$ and $x_k \notin Y$ but $x_j \in Y$ for $k < j \leq n+1$. Since $x_{k+1}, \ldots, x_{n+1} \in Y$ and the elements $x_1, \ldots, x_{n+1}$ and $y_1, \ldots, y_n$ both appear in order of nondecreasing deadlines, it must be the case that $d(y_n) \geq d(x_{n+1})$ and, in general, $d(y_i) \geq d(x_{i+1})$, $k \leq i \leq n$. Also, $d(x_i) \geq i$, $1 \leq i \leq n+1$. Thus, $d(y_i) \geq d(x_{i+1}) \geq i+1$, $k \leq i \leq n$. This implies that in the schedule $y_1, \ldots, y_n$ we can "shift" the elements $y_k, \ldots, y_n$ so that they now complete at times $k+1, \ldots, n+1$, respectively and are all still on time. This leaves a "gap" at time k. Since $x_k \notin Y$ and $d(x_k) \geq k$, we move x_k into this gap. We now have the set $Y + x_k$ with feasible schedule $y_1, \ldots, y_{k-1}, x_k, y_k, \ldots, y_n$. Therefore, $Y + x_k$ is a feasible set and the exchange property is satisfied. □

Since we have shown that M_S is a matroid and that the scheduling problem can be formulated as that of finding a basis of maximum weight in this matroid, the matroid greedy

algorithm can be applied to solve this problem. Specifically, the matroid greedy algorithm begins by sorting the jobs in order of nonincreasing weights. The set X is initially empty. Each job e_i is considered according to the sorted order and is added to X if and only if $X + e_i$ is independent. To test if $X + e_i$ is independent, all the jobs in $X + e_i$ are sorted in order of nondecreasing deadlines. Each job in this sorted order is then checked to determine if it completes by its deadline. If all the jobs are completed by their deadlines, $X + e_i$ is independent and e_i is added to X. Otherwise, the algorithm does not add e_i to X. When all jobs have been considered, the set X is a feasible set of maximum total weight. To find an optimal schedule, we simply sort the elements in X in order of nondecreasing deadlines. We then append the elements in $S - X$ in arbitrary order to the end of this schedule. Exercise 8 provides a small example on which the algorithm may be performed.

A Task Assignment Problem

A company has m employees $E = \{e_1, \ldots, e_m\}$ and n tasks $T = \{t_1, \ldots, t_n\}$ that must be completed. Each employee is qualified to perform a certain subset of these tasks but has time to perform at most one task. There is a positive real weight $w(t_i)$ associated with each task, representing the value or priority of that task. The objective is to find an assignment of tasks to employees, where each task is assigned to at most one employee and each employee is assigned to at most one task, such that the value of the completed tasks is maximized.

This optimization problem can be modelled with a *bipartite graph* in which there are two vertex sets $E = \{e_1, \ldots, e_m\}$ and $T = \{t_1, \ldots, t_n\}$. There is an edge from a vertex e_i to a vertex t_j if employee e_i is qualified to perform task t_j. A *matching* in the graph is a subset of edges such that no two edges share a common vertex. For a given matching, a vertex is said to be *matched* if some edge in the matching is incident on it. Our objective then is to find a matching in the graph that maximizes the sum of the weights of the matched vertices in T.

This problem too can be solved with a greedy algorithm. To do so we begin by defining a matroid known in the literature as a *transversal matroid*. Given a bipartite graph with vertices E and T we define $X \subseteq T$ to be *matchable* if there exists some matching in the graph that matches every vertex in X to some vertex in E.[3] Let $\mathcal{I}$ denote the set of all matchable subsets of T. We claim that $M_T = (T, \mathcal{I})$ is a matroid. Notice that an optimal solution to our problem is exactly that of finding an independent set of maximum weight in the matroid. Since all tasks in T have positive weight, an independent set of maximum weight is a basis of maximum weight. Thus, by showing that M_T is a matroid we will be able to solve the optimization problem using the matroid greedy algorithm.

Lemma 4 *Given a bipartite graph with vertices $E \cup T$, let $\mathcal{I}$ denote the set of all matchable subsets of T. Then $M_T = (T, \mathcal{I})$ is a matroid.*

Proof: The heredity property is satisfied because $\emptyset$ is trivially matchable and any subset of a matchable set is clearly also matchable.

[3]The technical name for such a subset is a *partial transversal*.

To show that the exchange property is satisfied, let X and Y be two elements of $\mathcal{I}$ such that $|X| > |Y|$. Since X and Y are matchable, let M_X and M_Y be matchings that match the vertices in X and Y, respectively, with vertices in E. Color the edges of $M_X - M_Y$ black, the edges of $M_Y - M_X$ white , and the edges of $M_X \cap M_Y$ gray. Notice that every edge in $M_X \cup M_Y$ is colored black, white, or gray. Since the number of edges in M_X is exactly equal to the number of vertices in X, and similarly for M_Y and Y, $|M_X| > |M_Y|$. This implies that there are more black edges than white edges.

Next, consider the subgraph M induced by the black and white edges.[4] Observe that each vertex in M is incident on at most two edges in M because at most it is incident on one edge in M_X and one edge in M_Y. Therefore, the vertices of M have degree one or two. This implies that M comprises only cycles and paths. Because no vertex of M can be incident on two edges of the same color, the edges on these cycles and paths alternate between black and white; these are called *alternating* cycles and paths. Each alternating cycle has an equal number of black and white edges. Since there are more black edges than white edges, some alternating path must have more black edges than white edges.

Let P be an alternating path with more black edges than white edges. Let $v_1, v_2, \ldots, v_k$ denote the vertices on path P from one endpoint to the other. Since P has more black edges than white edges, the first and last edges on this path must be black. Therefore, the path has an odd number of edges and thus an even number of vertices. Since the graph is bipartite, the vertices alternate between being in E and T. This means that one endpoint of P is in E and the other is in T. Without loss of generality, assume that $v_1 \in T$.

We now claim that $v_1 \in X - Y$. First, $v_1 \in X$ because it is incident on an edge in M_X. Assume, by way of contradiction, that $v_1 \in Y$. Then M_Y must contain an edge incident on v_1. Such an edge is either white or gray. The black edge in P incident on v_1 is, by definition, from matching M_X. Since there cannot be a second edge of M_X incident on v_1, no gray edge is incident on v_1. If there is a white edge incident on v_1 then this edge is part of P, contradicting the assumption that v_1 is an endpoint of P. Thus, $v_1 \in X - Y$.

Since $v_1 \in X - Y$, we now consider the set $Y + v_1$. To demonstrate that $Y + v_1$ is matchable, consider the matching M_Y modified by removing the white edges in P from M_Y and adding the black edges in P to M_Y. This set is still a matching and, in addition to matching every vertex in Y, matches v_1 as well. Thus $Y + v_1$ is matchable and is therefore an element of $\mathcal{I}$. □

Now that we have a matroid corresponding to this matching problem, we can use the matroid greedy algorithm to determine an optimal solution: for any graph, we first sort the vertices in T in nonincreasing order of weight and then we repeatedly choose a maximum weight vertex that maintains independence. The problem of determining if a set is independent in this context, that is whether a subset of T is matchable, can be solved using the bipartite matching algorithm often covered in a graph theory course (Bondy & Murty, 1976) or network flow algorithms generally covered in an algorithms course (Cormen *et al.*, 1990). Exercise 9 provides an example on which the greedy algorithm may be performed. This example is sufficiently small that testing for independence can easily be performed by inspection.

[4]The subgraph *induced* by the black and white edges is the graph formed by considering only those edges and the vertices incident to them.

Conclusion

We have seen that matroids are a powerful tool for constructing and showing the correctness of greedy algorithms. Matroids can also be used to develop efficient algorithms for even more difficult optimization problems where simple greedy algorithms fail. For example, consider a variant of the bipartite matching problem that arose in the task assignment problem. In that problem, weights were associated with a subset of the vertices in the graph and the objective was to find a matching that maximized the sum of these weights. Now consider the situation in which there is a positive weight associated with each edge, rather than with some vertices, and we wish to find a matching that maximizes the sum of the weights on the matched edges. Although the greedy algorithm does not always find an optimal solution for this problem, the problem can be solved using the notion of *matroid intersections*.

Given two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ over the same set E and a weight function $w : E \to \mathbb{R}$, the matroid intersection problem for two matroids is that of finding an element $X \in \mathcal{I}_1 \cap \mathcal{I}_2$ of maximum weight. Efficient algorithms are known for solving the matroid intersection problem for two matroids. Many discrete optimization problems, including the aforementioned bipartite matching problem with edge weights, can be formulated as a matroid intersection problem for two matroids.

Some optimization problems that cannot be solved by greedy algorithms or even using the intersection of two matroids can be formulated as the intersection of three or more matroids. The traveling salesperson problem, for example, can be formulated as that of finding an independent set of maximum weight in three matroids. Unfortunately, the matroid intersection problem for three or more matroids is NP-complete.

Several mathematical structures related to matroids have also been studied. For example, an *antimatroid* is a structure with a weaker version of the heredity property but a stronger version of the exchange property. A common framework for matroids and antimatroids is a structure called a *greedoid* (Korte *et al.*, 1991), which uses the weaker versions of both the heredity and the exchange properties. These two structures have applications in the development of greedy algorithms for optimization problems as well.

Finally, while we have investigated applications of matroids to discrete optimization, matroids and their related structures have a variety of other applications in mathematics. Active areas of current research are in applications of such structures in algebra, geometry, and topology.

We refer the interested reader to several excellent books. Oxley's text provides a comprehensive introduction to matroid theory. Texts by Lawler and by Papadimitriou and Steiglitz discuss the applications of matroids to discrete optimization.

References

Bondy, J. A., & Murty, U. S. R. 1976. *Graph theory with applications*. London: Macmillan.

Borůvka, O. 1926. On a certain minimal problem. *Prace moravske predovedecke spolecrosti*, **3**, 37–58.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. 1990. *Introduction to algorithms*. New York: McGraw-Hill.

Garey, M. R., & Johnson, D. S. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman and Company.

Korte, B., Lovász, L., & Schrader, R. 1991. *Greedoids*. Berlin: Springer Verlag.

Kruskal, J. B. 1956. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proceedings of the american mathematical society*, **7**, 48–50.

Lawler, E. 1976. *Combinatorial optimization: Networks and matroids*. New York: Holt, Rinehart, and Winston.

Oxley, J. G. 1992. *Matroid theory*. New York: Oxford University Press.

Papadimitriou, C., & Steiglitz, K. 1982. *Combinatorial optimization: Algorithms and complexity*. Englewood Cliffs, NJ: Prentice Hall.

Prim, R. C. 1957. The shortest connecting network and some generalisations. *Bell systems technical journal*, **36**, 1389–1401.

Whitney, H. 1935. On the abstract properties of linear independence. *American journal of mathematics*, **57**, 509–533.

Exercises

1. Prove Lemma 1.

2. Let $M = (E, \mathcal{I})$ be a matroid. Analogous to the definition in a vector space, the *rank* of a set $X \subseteq E$, denoted $r(X)$, is the size of a largest independent subset of X. Let B be a basis of M. Show that $r(B) = r(E) = |B|$.

3. Recall that if V is a vector space and $X \subseteq V$, the *span* of X, denoted $\text{span}(X)$, is defined to be the subspace of all vectors that can be expressed as a linear combination of vectors in X. One property of vector spaces is that for any basis B in vector space V, $\text{span}(B) = V$.

   If $M = (E, \mathcal{I})$ is a matroid and $X \subseteq E$, the span of X, denoted $\text{span}(X)$, is defined to be a maximal superset Y of X such that $r(Y) = r(X)$. Let $M = (E, \mathcal{I})$ be a matroid and let B be a basis in the matroid. Show that $\text{span}(B) = E$.

4. In the second paragraph of this paper we described a problem involving a weighted graph in which each vertex corresponds to a city, each edge corresponds to an existing phone line, and the weight on each edge corresponds to the cost of leasing that phone line. We assumed that the weight on each edge is a positive real number. We then argued that the problem of finding the least expensive subset of edges that permit us to find a path between any two vertices is exactly that of finding a minimum spanning

tree in the graph. Show that if the edge weights are not necessarily all positive then a minimum spanning tree does not necessarily give an optimal solution to this problem.

5. Does the matroid greedy algorithm find a minimum spanning tree if edges can have arbitrary real weights?

6. In the scheduling problem we assumed that all jobs had positive weights. Can the matroid greedy algorithm be used to solve this problem if jobs can have arbitrary real weights?

7. Consider the weighted graph shown in Figure 4. Use the greedy algorithm described earlier to find a minimum spanning tree in this graph.

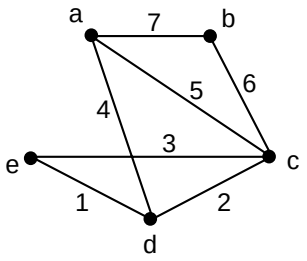

Figure 4: Weighted graph for Exercise 7.

8. Consider the scheduling problem for five jobs with deadlines $d(1) = 2$, $d(2) = 3$, $d(3) = 1$, $d(4) = 2$, $d(5) = 4$ and weights $w(1) = 3$, $w(2) = 1$, $w(3) = 2$, $w(4) = 5$, $w(5) = 1$.

   (a) Use the greedy algorithm to find an optimal schedule.

   (b) What is the sum of the weights of the on time jobs in an optimal schedule in this case?

   (c) How does this compare to the sum of the weights of the on time jobs in the schedule $1, 2, 3, 4, 5$?

9. Consider the task assignment problem for a set of three employees, $\{e_1, e_2, e_3\}$, and four tasks, $\{t_1, t_2, t_3, t_4\}$. Employee e_1 is qualified to perform task t_4, employee e_2 is qualified to perform tasks t_1 and t_2, and employee e_3 is qualified to perform tasks t_1 and t_3. The weights on the tasks are $w(t_1) = 5$, $w(t_2) = 4$, $w(t_3) = 2$, and $w(t_4) = 1$.

   (a) Use the greedy algorithm fo find an optimal solution for this instance of the task assignment problem.

   (b) What is the total weight of the completed tasks in this solution?

10. Let E be a set, and let $P_1, P_2, \ldots, P_k$ be a *partition* of E. That is, $E = P_1 \cup P_2 \cup \ldots \cup P_k$ and for all i, j, $1 \leq i < j \leq k$, $P_i \cap P_j = \emptyset$. A set S is said to *partially represent*

$P_1, \ldots, P_k$ if S contains at most one member of each of $P_1, \ldots, P_k$. Let $\mathcal{I}$ be the collection of sets which partially represent $P_1, \ldots, P_k$. In this exercise you will verify that $M_P = (E, \mathcal{I})$ is a matroid.

(a) Show that M_P satisfies the heredity property.

(b) Show that M_P satisfies the exchange property.

11. In this problem we investigate another discrete optimization problem and its corresponding matroid. Assume that there are n computers in a network. Ideally, every computer in the network will be paired up with exactly one other computer so that the two computers can exchange their files periodically for backup purposes. For compatibility reasons, only certain pairs of computers can be matched and it therefore may not be possible to find a match for every computer. Each computer administrator has stipulated a fee that they are willing to pay to have their computer matched. The company that provides the matching service would like to find a matching that maximizes the total fees that it can collect.

    This optimization problem can be modelled as a graph in which vertices correspond to computers, edges correspond to computers that can potentially be matched, and the weight on each vertex represents the fee that will be paid if that vertex is matched. Recall that a *matching* in a graph is defined to be a set of edges of G such that no two edges share a common vertex. Our objective is to find a matching that maximizes the sum of the weights on the matched vertices.

    Notice that this graph matching problem is slightly different from the one that arose in the task assignment problem. First, the graph that models this problem is not necessarily bipartite. In addition, in this problem every vertex has an associated weight.

    To solve this problem we consider a new matroid called a *matching matroid*. Let $G = (V, E)$ be a graph. Define $X \subseteq V$ to be *matchable* if there exists some matching $M \subseteq E$ such that every vertex in X is incident on some edge in M. For convenience, we say that an edge in M *covers* its endpoints or M covers the set X. Unlike the case of the bipartite matchings in the task assignment problem, an edge in matching M may be incident on zero, one, or two vertices in X. Let $\mathcal{I}$ be the set of all matchable subsets of V. Assume that we already have an algorithm that determines whether or not a set is matchable and, if it is matchable, the algorithm constructs a matching for it.

    (a) If $M_M = (V, \mathcal{I})$ is a matroid and w is a weight function from the vertices to the positive reals, show that a basis of maximum weight in M_M corresponds to an optimal solution to this optimization problem.

    (b) We will now show that M_M is a matroid in a sequence of steps. Begin by showing that M_M satisfies the heredity property.

    (c) To show that the exchange property is satisfied, begin by considering $X, Y \in \mathcal{I}$ such that $|X| > |Y|$ and consider matchings M_X and M_Y that cover X and Y, respectively. Argue that if M_Y covers a vertex in $X - Y$ then the exchange property is satisfied.

(d) Now consider the case that M_Y covers no vertex in $X - Y$. As in the proof of Lemma 4, color the edges in $M_X - M_Y$ black, color the edges in $M_Y - M_X$ white, and color the edges in $M_X \cap M_Y$ gray. Show that the gray edges cover at least as many vertices in Y as in X.

(e) Recall that an *alternating path* and an *alternating cycle* is a path and cycle, respectively, that alternates between black and white edges. Show that the edges on the alternating cycles cover at least as many vertices in Y as in X.

(f) Show that the alternating paths contain more vertices of X than Y.

(g) Show that an endpoint of an alternating path cannot be in both X and Y.

(h) Show that there exists an alternating path $v_1, \ldots, v_k$ such that $v_1 \in X - Y$ and $v_k \notin Y$.

(i) Show that $Y + v_1 \in \mathcal{I}$ by showing that a matching exists that covers this set. Conclude that M_M is a matroid.

12. As an example of the optimization problem described in Exercise 11, consider a network comprising five computers, a, b, c, d, e, represented by the graph in Figure 5. An edge between two vertices indicates that the corresponding computers are compatible and may be matched to one another. The number next to each vertex represents the fee (or weight) associated with that computer. Use the greedy algorithm to find an optimal solution for this instance of the optimization problem.

Figure 5: Graph for Exercise 12.

Solutions to the Exercises

1. Let $M = (E, \mathcal{I})$ be a matroid and let X and Y be two bases of M. If $|X| > |Y|$ then, by the exchange property, there exists some $e \in X - Y$ such that $Y + e \in \mathcal{I}$. This contradicts the assumption that Y is a basis. By symmetry, it cannot be the case that $|X| < |Y|$. Thus $|X| = |Y|$, proving the lemma.

2. Since B is an independent subset, $r(B) = |B|$. Further, note that $r(E) = |C|$ where C is some basis of the matroid. From the previous problem, B and C must have the same cardinality, so $r(E) = |B| = r(B)$.

3. From the previous problem $r(B) = r(E)$. Since E is a maximal superset of B, the claim follows.

4. If some edge weights are negative, it may be advantageous to select a subset of edges that contain a cycle. For example, consider a graph with three vertices a, b, and c and edges ab, bc, ca with weights -1, -2, -3, respectively. In this case, we would select all three edges in spite of the fact that they form a cycle.

5. Yes. We wish to find a basis of maximum weight regardless of whether the weights are positive or negative. The matroid greedy algorithm was shown to work for any matroid $M = (E, \mathcal{I})$ with weight function $w : E \to \mathbb{R}$.

6. No. We showed that an optimal solution to this problem corresponds to a basis of maximum weight in the matroid assuming that the weights are positive. If the weights are not positive this claim is not true. In particular, a basis does not necessarily correspond to an optimal solution to this problem when some jobs have negative weights. Therefore, the matroid greedy algorithm cannot be used for this problem if negative weights are permitted.

7. The algorithm selects the edges with weights 1, 2, 4, and 6 to be in the spanning tree.

8.

   (a) The greedy algorithm finds schedule $4, 1, 2, 5, 3$ or $1, 4, 2, 5, 3$, depending on how the tie between jobs 1 and 4 is broken.

   (b) The sum of the weights of the on time jobs is 10.

   (c) The sum of the weights of the on time jobs in schedule $1, 2, 3, 4, 5$ is 4.

9.

   (a) The algorithm selects tasks t_1, t_2, and t_4. These tasks can be assigned to employees e_3, e_2, and e_1, respectively.

   (b) The total weight of this solution is 10.

10.

   (a) The empty set has no elements from any P_i, $1 \leq i \leq k$, and is therefore independent. Let $X \in \mathcal{I}$ and $Y \subseteq X$. Since X has at most 1 element from each P_i, $1 \leq i \leq k$, Y does as well. Hence, Y is independent.

   (b) Let $X, Y \in \mathcal{I}$ with $|X| > |Y|$. There must exist some i, $1 \leq i \leq k$, such that X contains an element e of P_i but such that Y contains no element of P_i. Therefore, $e \in X - Y$ and $Y + e \in \mathcal{I}$.

11.

   (a) Each set in $\mathcal{I}$ corresponds to a solution to the problem. Since the weight function is positive, an optimal solution will be a set in $\mathcal{I}$ such that the addition of any other element of V will result in a set not in $\mathcal{I}$. This is a basis of $\mathcal{I}$.

(b) The hereditary property is satisfied since $\emptyset \in \mathcal{I}$ and if $X \in \mathcal{I}$ is covered by some matching M, then M also covers any subset $Y \subseteq X$ and thus $Y \in \mathcal{I}$.

(c) If M_Y covers some vertex $v \in X - Y$ then M_Y covers $Y + v$ and thus $Y + v \in \mathcal{I}$ and the exchange property is satisfied.

(d) Assume that M_Y covers no vertex in $X - Y$. The gray edges are those in $M_X \cap M_Y$ and thus cannot cover a vertex in $X - Y$. Thus, if a gray edge covers a vertex in X that vertex must be in Y as well. Therefore, the gray edges cover at least as many vertices in Y as in X.

(e) Every vertex on an alternating cycle is covered by an edge in M_Y. Thus, any vertex in X on an alternating cycle must also be in Y. Therefore, the alternating cycles cover at least as many vertices in Y as in X.

(f) Every vertex in $X \cup Y$ is covered by a gray, black, or white edge. From the previous two results, we know that the gray edges and the alternating cycles contain at least as many vertices in Y as in X. Since $|X| > |Y|$, the alternating paths must contain more vertices of X than Y.

(g) Let $v_1, \ldots, v_k$ be an alternating path and assume that $v_1 \in X \cap Y$. Then both M_X and M_Y contain edges incident on v_1. Since edge (v_1, v_2) is from either M_X or M_Y but not both, there must exist another edge from M_X or M_Y incident on v_1. Thus, v_1 is either incident on two edges from the same matching or is not an endpoint of the alternating path, both of which are contradictions.

(h) From the above observations, there must exist an alternating path P with more vertices in X than Y. Let $v_1, \ldots, v_k$ be the vertices of P from one endpoint to the other. At least one endpoint must be in $X - Y$ since otherwise the path has at least as many vertices in Y as in X. Without loss of generality, assume $v_1 \in X - Y$. If $v_k \in Y$ then, from the above observation, it cannot be in X. In this case, the path contains at least as many vertices in Y as in X, a contradiction.

(i) Let us remove the white edges in P from M_Y and add the black edges of P to M_Y. This is still a matching and this matching is incident on all vertices in $Y + v_1$. This confirms that $Y + v_1$ is matchable and that M_M is a matroid.

12. An optimal solution comprises vertices e, b, d, c. The total weight of this solution is 14 and a matching for this set of vertices matches e with c and matches b with d.

About the Authors

Christian Jones completed a B.S. in mathematics from Harvey Mudd College and is currently pursuing a Ph.D. in mathematics at the University of Florida. His research interests are in combinatorics, especially algorithms in graph theory, matroid theory, and number theory.

Ran Libeskind-Hadas completed a B.S. in applied mathematics from Harvard University and M.S. and Ph.D. in computer science from the University of Illinois at Urbana-Champaign. He is an associate professor of computer science at Harvey Mudd College. His research interests

are in algorithms and parallel computing and he teaches courses in discrete mathematics, algorithms, and other areas of computer science.