Greedy Algorithms

In-Class Exercise: Fractional Knapsack

- You are given a collection of n items.
- Each item i has a weight w_i and a value v_i
- You have a bag that can hold a total weight of W
- You want to maximize the value of the items in your bag (assume you can take fractional items)
- Design an algorithm to decide which items to pick.

and how much of each

$A = \{a_1, \ldots, a_n\}$

$O = \{o_1, \ldots, o_n\}$

$a_i \leq o_i$ for all i

or $o_i \leq a_i$ for all i

In-Class Exercise: Fractional Knapsack

1. Sort items in desc. order of $\frac{v_i}{w_i}$
2. Add each item in order
3. When we reach the end, if we can't take all of the next item, take as much as possible

$\boxed{\text{FK algo}}$

Proof of Correctness of Frac. Knapsack Alg.

Claim: FK always finds the optimal solution

Proof: We will prove this using the greedy exchange method. Let items $1 \ldots n$ be the available items, sorted in desc. order of value/weight.

Suppose FK finds solution $A = \{a_1, \ldots, a_n\}$, where a_i represents how much of item i was taken.

Proof of Correctness of Frac. Knapsack Alg.

Let the optimal solution $O = \{o_1, \ldots, o_n\}$ represent how much of each item is taken in the optimal case.

We want to show that the total value of A is equal to the total value of O. Suppose for a contradiction that $Value(A) < Value(O)$.

Then A and O must differ somewhere.

Proof of Correctness of Frac. Knapsack Alg.

One thing to note is that there must exist an i where $a_i > o_i$ and there be a j where $o_j > a_j$. If this were not the case, then one solution would be a proper subset of the other, ~~thi~~ but because they both fill the bag to capacity, this is not possible.

Proof of Correctness of Frac. Knapsack Alg.

Because of how FK works, A must have the form $\{1, \cdots, 1, x, 0, \cdots, 0\}$, where $0 \leq x \leq 1$. From previous slide, there is some item i with $a_i > o_i$. Because both solutions have the same total weight (W), the extra weight taken by A on item i must be made up for by O taking more of other items. But because of the structure of A, those items

Proof of Correctness of Frac. Knapsack Alg.

for which O takes more must come after i. But because of how we sorted the items, those items after i necessarily have a value/wt ratio no more than i has. So if we remove an appropriate amount of weight from those later items in O, and increase the weight of item i in O, the total value of O will not decrease. So by

Proof of Correctness of Frac. Knapsack Alg.

iteratively applying this argument, we can transform O into A with no reduction in total value. Thus, the total value of A is equal to the total value of O, so A is optimal. □

In-Class Exercise: Knapsack

- Same as before, except you *cannot* take fractional items.
- Does your algorithm always find the optimal solution? Explain why, or give a counterexample.

$W = 4$

v	w	v/w
2	2	1
2	2	1
3.1	3.0	$3.1/3 > 1$

$4 = OPT$

$3.1 = Fk$

Proving Correctness of Greedy Algorithms

- ”Greedy stays ahead” interval scheduling
  1. Label your solution A = $\{a_1, \ldots, a_k\}$, label the optimal solution O = $\{o_1, \ldots, o_m\}$. (Note, the ordering of these elements depends on your particular problem.)
  2. Define a quality measure of how good a solution (or partial solution) is
  3. Prove that the greedy method is always at least as good as the optimal solution for all indices *r*

Proving Correctness of Greedy Algorithms

- "Greedy exchange" Kruskal's MST algo.
  1. Label your solution A = $\{a_1, \ldots, a_k\}$, label the optimal solution O = $\{o_1, \ldots, o_m\}$. (Note, the ordering of these elements depends on your particular problem.)
  2. Find a way in which your solution A differs from the optimal solution
  3. Show that if you swap the different element from A into O, the solution does not decrease in quality
  4. Repeat until O is converted into A; thus showing that A is at least as good as O

A Class of Greedy Algorithms

- Using *matroid theory*, we can prove that for a certain type of problem, a certain greedy algorithm will *always* be optimal

When Does a Problem Have a Greedy Solution? Some Background

set of sets

solution components

- A **subset system** is a finite set E and a weighted set I, where each element in I is a subset of E $I = \{\{3,4\},$
  - Example: $E = \{1, 2, 3, 4\}$, $I = \{\{1, 2\}, \{1\}, \{2, 4\}\}$ $\{3\}$ $\emptyset$ $\{4\}\}$
- A subset system is **hereditary** (or **closed under inclusion**) if for each element Y in I, if X is a subset of Y, then X is also in I

  set

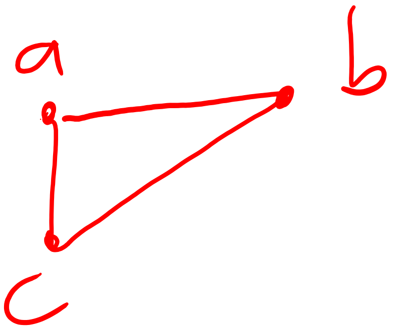
  - What elements would you need to add to the example above to make it closed under inclusion?

$I = \{\{1,2\}, \{1\}, \{2,4\}, \{2\}, \{\}, \{4\}\}$

Some examples

fixed G

1. Let E be the set of edges in a graph, and the elements of I are all sets of acyclic edges.
2. Let E be the set of edges in a graph, and the elements of I are the sets of edges that don't have any vertices in common



$I = \{\{\}, \{(a,b)\}, \{(b,c)\}, \{(a,c)\}, \{(a,b),(b,c)\}, \{(a,c),(b,c)\}, \{(a,b),(a,c)\}\}$

Some examples

Graph G

I contains all acyclic sets of edges

if $A \in I$ (in other words, A is an acyclic set of edges), and $B \subseteq A$, is $B \in I$?

yes!

because if A is acyclic, and B is a subset of A, B must also be acyclic, so $B \in I$.

Some examples

Graph G

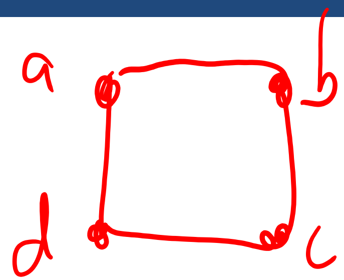
I contains all acyclic sets of edges

if $A \in I$ (in other words, A is an acyclic set of edges), and $B \subseteq A$, is $B \in I$?

yes!

because if A is acyclic, and B is a subset of A, B must also be acyclic, so $B \in I$.

Some examples



I = set of all sets of edges that don't share vertices

$I = \{\emptyset, \{(a,b)\}, \{(a,d)\}, \{(b,c)\}, \{(d,c)\},$
$\{(a,b),(d,c)\}, \{(a,d),(b,c)\}\}$

Is I closed under inclusion? yes

If A is a set of edges that don't overlap, and $B \subseteq A$, none of the edges in B can overlap, so $B \in I$.

Optimization Problems

system

- An optimization problem for a subset is a problem such that:

how much does it contribute to solution?

  - Input: elements in *E* and their weights, function to determine if a subset of *E* is in *I*
  - Output: a set *X* in *I* such that *X* has at least as much total weight as any set in *I*
  - Note: *I* might be very large, so we wouldn't want to just check every element. This is why we have a function to check if a subset is in *I*, rather than having *I* itself.

A Generic Greedy Algorithm

Input: elements in *E* and their weights, set *I* that is closed under conclusion (or a function that determines if a subset *X* of *E* is in *I*)

1. Sort the elements of *E* in descending order of weight
2. Set *X* to be the empty set
3. Traverse the list of elements in *E*, and in each step, add the current element as long as *X* will still be in *I*

Matroids

closed under inclusion

A hereditary subset system is a matroid if:

1. The empty set is in I
2. It is closed under inclusion
3. It satisfies the exchange property:

valid partial solutions

- If s and s' are elements of I, and s has fewer elements than s',

s s'

- Then there is some element e in $s' \setminus s$

solution component

- Such that $s + e$ is in I

Showing that a Problem is a Matroid

1. Define your sets *E* and *I*
   - *E* is the set of elements- these are usually objects that you add to the solution one at a time
   - *I* is the set of allowable solutions (e.g., those that do not violate some constraint)
2. Show that *I* is closed under inclusion
3. Show that *I* satisfies the exchange property

Putting it Together

Matroid Theorem: The generic greedy algorithm solves the optimization problem if and only if the subset system is a matroid.

Cardinality Theorem: A subset system is a matroid if and only if for any set A in E, any two maximal independent subsets of A have the same number of elements.

Proof of the Matroid Theorem

Matroid Theorem: The generic greedy algorithm solves the optimization problem for a hereditary subset system if and only if that hereditary subset system is a matroid.

Proof of the Matroid Theorem: Part 1

Matroid Theorem Part 1: The generic greedy algorithm solves the optimization problem for a hereditary subset system if that hereditary subset system is a matroid.

Proof: Suppose a hereditary subset system (*E, I*) is a matroid. Then it is closed under inclusion and satisfies the exchange property. Suppose *X* is the subset chosen by the greedy algorithm, and suppose *Y* is any other subset that has greater total weight than X, and $Y \in I$.

Proof of the Matroid Theorem: Part 1

Proof cont'd: Suppose a subset system (E, I) is a matroid. Then it satisfies the exchange property. Suppose X is the subset chosen by the greedy algorithm, and suppose Y is any other subset. with greater total weight that satisfies constraint.

X and Y must have the same size. (Why?) Exchange property.

Suppose for a contradiction that $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, where elements are listed in descending order of weight, and Y has greater weight than X.

Proof of the Matroid Theorem: Part 1

Proof cont'd: Suppose for a contradiction that $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$, where elements are listed in descending order of weight, and Y has greater weight than X.

earliest index

Let k be the ~~smallest value~~ such that $w(y_k) > w(x_k)$. (Why must there be such a k?)

Proof of the Matroid Theorem: Part 1

Proof cont'd: Let k be the smallest value such that $w(y_k) > w(x_k)$.

X : found by greedy

Z = α + {some element from β}

Y : better sol. than X

Define $\alpha = \{x_1, \ldots, x_{k-1}\}$, $\beta = \{y_1, \ldots, y_k\}$ (note the differences in indices!). By the exchange property, we can make a set Z by adding some element from β to α. All elements in β have weight ≥ $w(y_k)$, which is greater than $w(x_k)$, so Z has weight greater than $\{x_1, \ldots, x_k\}$. This means that the greedy algorithm skipped over an element when it added x_k! Contradiction.

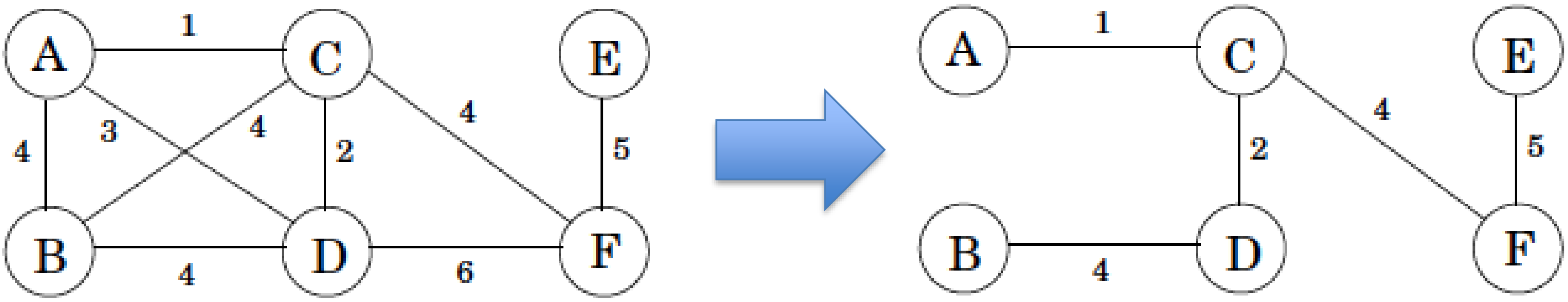
Thus, if your problem is a matroid, the GGA is optimal.

Putting it Together

- So what does this mean?
- If you can show that your problem corresponds to a matroid, you get a greedy algorithm for free, along with proof that it works!
- If your problem is not a matroid, does that mean you can't design a greedy algorithm?

Back to Kruskal's Algorithm

- Kruskal's Algorithm:
  1. Sort the edges from least cost to greatest cost
  2. Add each edge in order to the spanning tree, unless it would make a cycle



- How can we use matroid theory to prove correctness of Kruskal's Algorithm?

Back to Kruskal's Algorithm

- Kruskal's Algorithm matches the generic greedy algorithm described earlier
- If we want to prove that it is correct, we just need to show that the underlying problem is a matroid

Is the MST Problem a Matroid?

- What is *E*? (What are the objects that we add one at a time to the greedy solution?)

edges in graph

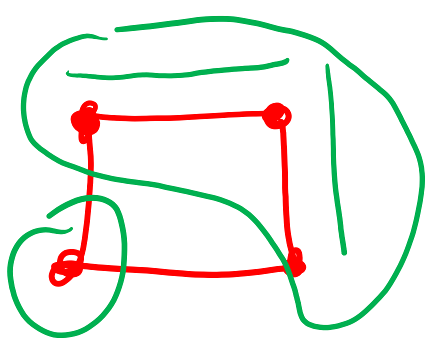
- What is *I*? (Which solutions/partial solutions are valid?) Is *I* closed under inclusion? yes

set of all ~~sets~~ acyclic sets of edges

Is the MST Problem a Matroid?

- Does *I* satisfy the exchange principle? Suppose *s* and *s'* are in *I*, and *s'* has more elements than *s*. Can we take an element from *s'*/*s* and add it to *s*, while still remaining in *I*? yes
- Suppose you have an acyclic set of *k* edges. How many trees are in the graph? n nodes

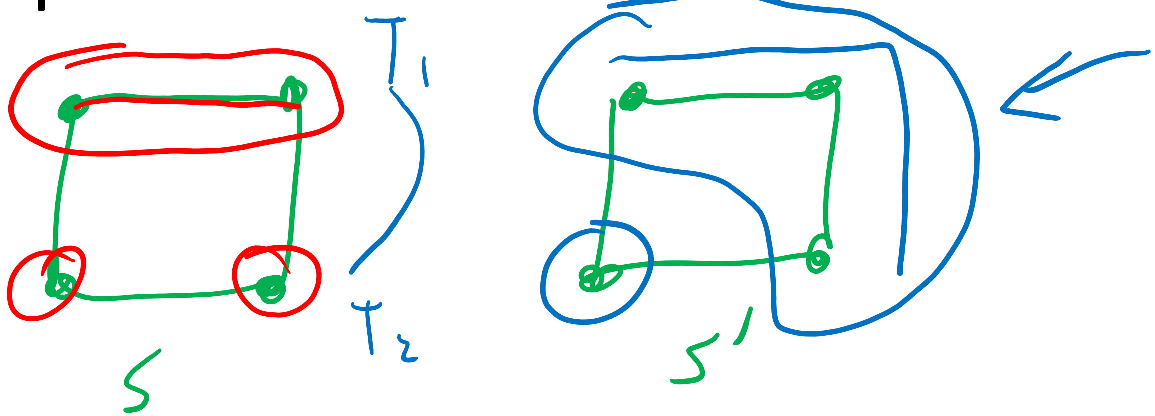
n - k



Is the MST Problem a Matroid?

might be single nodes

- If $|s| = k$, and $|s'| > k$, then s must produce more trees in the graph than s'.
- Thus, there is some tree in s' that contains vertices from multiple trees in s. Call these trees in s T_1 and T_2.



Is the MST Problem a Matroid?

- This means that *s'* contains an edge linking the T_1 nodes to the T_2 nodes.
- This edge can be added to *s* without adding a cycle, so the exchange property is satisfied.
- Thus, the MST problem is a matroid, so Kruskal's Algorithm is correct.