Greedy Algorithms and Matroids

Andreas Klappenecker

Giving Change

Coin Changing

Suppose we have n types of coins with values

v[1] > v[2] > … > v[n] > 0

Given an amount C, a positive integer, the following algorithm tries to give change for C:

```
m[1] := 0; m[2]:=0; … ; m[n] := 0; // multiplicities of coins
for i = 1 to n do
    while C > v[i] do
        C := C – v[i]; m[i]++;
    od;
od;
```

Coin Changing

Suppose that v[1] = 5; v[2] =2; v[1] = 1;

Let C = 14.

Then the algorithm calculates

m[1] = 2 and m[2] = 2

This gives the correct amount of change:

C = v[1]*m[1]+v[2]*m[2] = 5*2+2*2 = 14

Correctness

Suppose we have n types of coins with values

v[1] > v[2] > … > v[n] > 0

Input: A positive integer C

```
m[1] := 0; m[2]:=0; … ; m[n] := 0; // multiplicities of coins
for i = 1 to n do
    while C > v[i] do
        C := C – v[i]; m[i]++;
    od;
od;
```

// When is the algorithm correct for all inputs C?

Optimality

Suppose we have n types of coins with values

v[1] > v[2] > … > v[n] = 1

Input: A positive integer C

```
m[1] := 0; m[2]:=0; … ; m[n] := 0; // multiplicities of coins
for i = 1 to n do
    while C > v[i] do
        C := C – v[i]; m[i]++;
    od;
od;
// Does the algorithm gives the least number of coins?
```

Example

v[1] = 5; v[2] = 2; v[3] = 1;

If C<2, then the algorithm gives a single coin, which is optimal.

If C<5, then the algorithm gives at most 2 coins:

C = 4 = 2*2 // 2 coins

C = 3 = 2+1 // 2 coins

C = 2 = 2 // 1 coins

In each case this is optimal.

If C >= 5, then the algorithm uses the most coins of value 5 and then gives an optimal change for the remaining value < 5. One cannot improve upon this solution. Let us see why.

Example

Any optimal solution to give change for C with

C = v[1]*m[1] + v[2]*m[2] + v[3] * m[3]

with v[1] = 5; v[2] = 2; v[3] = 1;

must satisfy

a) m[3] <= 1 // if m[3] >= 2, replace two 1's with 2.

b) 2*m[2] + m[1] < 5 // otherwise use a 5

c) If C >= 5*k, then m[1] >= k // otherwise solution violates b)

Thus, any optimal solution greedily chooses maximal number of 5s. The remaining value in the optimal value has to choose maximal number of 2s, so any optimal solution is the greedy solution!

Example 2

Let v[1]=4; v[2]=3; v[3]=1;

The algorithm yields 3 coins of change for

C = 6 namely 6=4+1+1

Is this optimal?

Example 2

Any optimal solution to give change for C with

C = v[1]*m[1] + v[2]*m[2] + v[3] * m[3]

with v[1] = 4; v[2] = 3; v[3] = 1;

must satisfy

a) m[3] <= 2 // if m[3] > 2, replace two 1's with 3.

b) We cannot have both m[3]>0 and m[2]>0, for otherwise we could use a 4 to reduce the coin count.

c) m[2]<4 // otherwise use 4 to reduce number of coins

Greedy will fail in general. One can still find an optimal solution efficiently, but the solution is not unique. Why? 9=3+3+3=4+4+1

How Likely is it Optimal?

Let N be a positive integer.

Let us choose integer a and b uniformly at random subject to N>b>a>1. Then the greedy coin-changing algorithm with

v[1]=b; v[2]=a; v[3]=1

gives always optimal change with probability

$8/3\ N^{-1/2} + O(1/N)$

as Thane Plambeck has shown [AMM 96(4), April 1989, pg 357].

Greedy Algorithms

Greedy Algorithms

The development of a greedy algorithm can be separated into the following steps:

1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.
3. Demonstrate that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice that we have made, we arrive at an optimal solution to the original problem.

Greedy-Choice Property

The greedy choice property is that a globally optimal solution can be arrived at by making a locally optimal (=greedy) choice.

Optimal Substructure

A problem exhibits optimal substructure if and only if an optimal solution to the problem contains within it optimal solutions to subproblems.

Greedy Algorithms

Greedy algorithms are easily designed, but correctness of the algorithm is harder to show.

We will look at some general principles that allow one to prove that the greedy algorithm is correct.

Matroids

Matroid

Let S be a finite set, and F a nonempty family of subsets of S, that is, $F \subseteq P(S)$.

We call (S,F) a matroid if and only if

M1) If $B \in F$ and $A \subseteq B$, then $A \in F$.

[The family F is called hereditary]

M2) If $A,B \in F$ and $|A|<|B|$, then there exists x in $B \setminus A$ such that $A \cup \{x\}$ in F

[This is called the exchange property]

Example 1 (Matric Matroids)

Let M be a matrix.

Let S be the set of rows of M and

F = { A | A⊆S, A is linearly independent }

Claim: (S,F) is a matroid.

Clearly, F is not empty (it contains every row of M).

M1) If B is a set of linearly independent rows of M, then any subset A of M is linearly independent. Thus, F is hereditary.

M2) If A, B are sets of linearly independent rows of M, and |A|<|B|, then dim span A < dim span B. Choose a row x in B that is not contained in span A. Then A∪

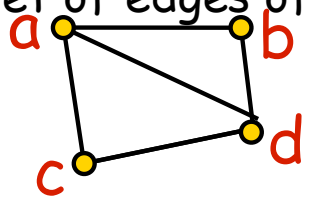
Undirected Graphs

Let V be a finite set,

E a subset of { e | e ⊆ V, |e|=2 }

Then (V,E) is called an undirected graph.

We call V the set of vertices and E the set of edges of the graph.



V = {a,b,c,d}

E = { {a,b}, {a,c}, {a,d}, {b,d}, {c,d} }

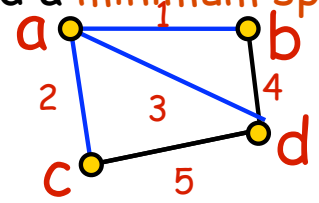
Induced Subgraphs

Let $G=(V,E)$ be a graph.

We call a graph (V,E') an induced subgraph of G if and only if its edge set E' is a subset of E.

Spanning Trees

Given a connected graph G, a spanning tree of G is an induced subgraph of G that happens to be a tree and connects all vertices of G. If the edges are weighted, then a spanning tree of G with minimum weight is called a minimum spanning tree.



Example 2 (Graphic Matroids)

Let $G=(V,E)$ be an undirected graph.

Choose $S = E$ and

$F = \{ A \mid H = (V,A)$ is an induced subgraph of G such that H is a forest $\}$.

Claim: (S,F) is a matroid.

M1) F is a nonempty hereditary set system.

M2) Let A and B in F with $|A| < |B|$. Then (V,B) has fewer trees than (V,A). Therefore, (V,B) must contain a tree T whose vertices are in different trees in the forest (V,A). One can add the edge x connecting the two different trees to A and obtain another forest $(V,A \cup \{x\})$.

Weight Functions

A matroid (S,F) is called weighted if it equipped with a weight function w: S->$\mathbf{R}^+$, i.e., all weights are positive real numbers.

If A is a subset of S, then

$$w(A) := \sum_{a \text{ in } A} w(a).$$

Weight functions of this form are sometimes called "linear" weight functions.

Greedy Algorithm for Matroids

Greedy(M=(S,F),w)

$A := \emptyset$;

Sort S into monotonically decreasing order by weight w.

for each x in S taken in monotonically decreasing order **do**

    if $A \cup \{x\}$ in F **then** $A := A \cup \{x\}$; **fi**;

od;

return A;

Correctness

Theorem: Let M= (S,F) be a weighted matroid with weight function w. Then Greedy(M,w) returns a set in F of maximal weight.

[Thus, even though Greedy algorithms in general do not produce optimal results, the greedy algorithm for matroids does! This algorithm is applicable for a wide class of problems. Yet, the correctness proof for Greedy is not more difficult than the correctness for special instance such as Huffman coding. This is economy of thought!]

Complexity

Let n = |S| = # elements in the set S.

Sorting of S: $O(n \log n)$

The for-loop iterates n times. In the body of the loop one needs to check whether $A \cup \{x\}$ is in F. If each check takes f(n) time, then the loop takes $O(n\, f(n))$ time.

Thus, Greedy takes $O(n \log n + n\, f(n))$ time.

Minimizing or Maximizing?

Let M=(S,F) be a matroid.

The algorithm Greedy(M,w) returns a set A in F maximizing the weight w(A).

If we would like to find a set A in F with minimal weight, then we can use Greedy with weight function

$$w'(a) = m - w(a) \qquad \text{for } a \text{ in } A,$$

where m is a real number such that $m > \max_{s \text{ in } S} w(s)$.

Matric Matroids

Let M be a matrix. Let S be the set of rows of the matrix M and

F = { A | A⊆S, A is linearly independent }.

Weight function w(A)=|A|.

What does Greedy((S,F),w) compute?

The algorithm yields a basis of the vector space spanned by the rows of the matrix M.

Graphic Matroids

Let $G=(V,E)$ be an undirected connected graph.

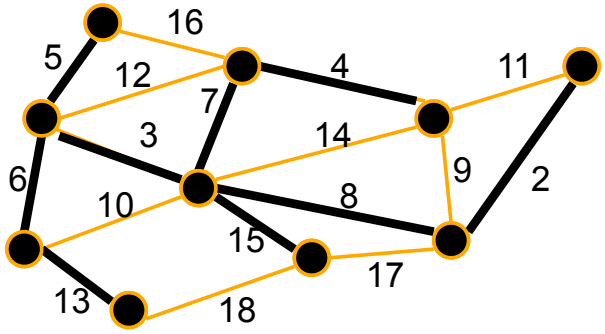
Let $S = E$ and $F = \{ A \mid H = (S,A)$ is an induced subgraph of G such that H is a forest $\}$.

Let w be a weight function on E.

Define $w'(a)=m-w(a)$, where $m>w(a)$, for all a in A.

Greedy((S,F), w') returns a minimum spanning tree of G.
This algorithm in known as Kruskal's algorithm.

Kruskal's MST algorithm



Consider the edges in increasing order of weight, add an edge iff it does not cause a cycle

[Animation taken from Prof. Welch's lecture notes]

Conclusion

Matroids characterize a group of problems for which the greedy algorithm yields an optimal solution.

Kruskals minimum spanning tree algorithm fits nicely into this framework.