CSE4403 3.0 & CSE6002E - Soft Computing
Winter Semester, 2011

Greedy algorithm

A greedy algorithm is any algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding the global optimum. For example, applying the greedy strategy to the traveling salesman problem yields the following algorithm: "At each stage visit the unvisited city nearest to the current city". In general, greedy algorithms are used for optimization problems.

Specifics
In general, greedy algorithms have five pillars:

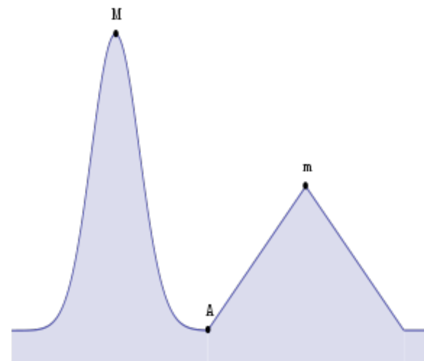
1. A candidate set, from which a solution is created
2. A selection function, which chooses the best candidate to be added to the solution
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
4. An objective function, which assigns a value to a solution, or a partial solution, and
5. A solution function, which will indicate when we have discovered a complete solution

Greedy algorithms produce good solutions on some mathematical problems, but not on others. Most problems, for which they work, will have two properties:

Greedy choice property
We can make whatever choice seems best at the moment and then solve the sub-problems that arise later. The choice made by a greedy algorithm may depend on choices made so far but not on future choices or all the solutions to the sub-problem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage's algorithmic path to solution.

Optimal substructure
A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems.

Cases of failure
Starting at A, a greedy algorithm will find the local maximum at "m", oblivious of the global maximum at "M"



For many other problems, greedy algorithms fail to produce the optimal solution, and may even produce the *unique worst possible* solution. One example is the traveling salesman problem mentioned above: for each number of cities there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour.

Imagine a coin example with only 25-cent, 10-cent, and 4-cent coins. The greedy algorithm would not be able to make change for 41 cents, since after committing to use one 25-cent coin and one 10-cent coin it would be impossible to use 4-cent coins for the balance of 6 cent. Whereas a person or a more sophisticated algorithm could make change for 41 cents change with one 25-cent coin and four 4-cent coins.

Types

Greedy algorithms can be characterized as being 'short sighted', and as 'non-recoverable'. They are ideal only for problems, which have 'optimal substructure'. Despite this, greedy algorithms are best suited for simple problems (e.g. giving change). It is important, however, to note that the greedy algorithm can be used as a selection algorithm to prioritize options within a search, or branch and bound algorithm. There are a few variations to the greedy algorithm:

Pure greedy algorithms

Orthogonal greedy algorithms

Applications

Greedy algorithms mostly (but not always) fail to find the globally optimal solution, because they usually do not operate exhaustively on all the data. They can make commitments to certain choices too early, which prevent them from finding the best overall solution later. For example, all known greedy coloring algorithms for the graph-coloring problem and all other NP-complete problems do not consistently find optimum solutions. Nevertheless, they are useful because they are quick to think up and often give good approximations to the optimum.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like dynamic programming. Examples of such greedy algorithms are Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees, Dijkstra's algorithm for finding single-source shortest paths, and the algorithm for finding optimum Huffman trees.

The theory of matroids, and the more general theory of greedoids, provide whole classes of such algorithms.

Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighboring node, which is "closest" to the destination. The notion of a node's location (and hence "closeness") may be determined by its physical location, as in geographic routing used by ad-hoc networks. Location may also be an entirely artificial construct as in small world routing and distributed hash table.

References

Introduction to Algorithms (Cormen, Leiserson, and Rivest) 1990, Chapter 16 "Greedy Algorithms" p. 329.

Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 "Greedy Algorithms".

G. Gutin, A. Yeo and A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. Discrete Applied Mathematics 117 (2002), 81–86.

J. Bang-Jensen, G. Gutin and A. Yeo, When the greedy algorithm fails. Discrete Optimization 1 (2004), 121–127.

G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems, Discrete Optimization 3 (2006), 288–298.