Approximation algorithms

The approximation algorithms are used when we want to find an approximate solution to a problem that can’t be solved in polynomial time. Unfortunately, most interesting discrete optimization problems are NP-hard.

P – a class of problem for which an algorithm can provide the optimum solution in polynomial time

NP – the class of problems for which an answer can be verified in polynomial time

Unless P = NP, there are no efficient algorithms to find optimal solutions to such problems. An old engineering slogan says: “Fast. Cheap. Reliable. Choose two.” This means that if P != NP, we can’t simultaneously have algorithms that (1) find optimal solutions (2) in polynomial time (3) for any instance. At least one of these requirements must be relaxed. The most common approach is to relax the requirement of finding an optimal solution and instead settle for a solution that’s good enough especially when it can be found in a second or less.

The **approximation ratio** (or **approximation factor**) of an algorithm is the ratio between the result obtained by the algorithm and the optimal cost or profit. Typically this ratio is taken in whichever direction makes it bigger than one. For example, an algorithm that solves for a cost of $2 an instance of a problem that has an optimal cost of $1 has approximation ratio 2; but an algorithm that sells 10 airplane tickets (a profit of 10) when the optimum is 20 also has approximation ratio 2.

The algorithm that I’ve implemented is:

C=∅  
E – the set of edges  
while E ≠ ∅

let (u,v) be an arbitrary edge of E

C = C ∪ {u,v}

remove from E every edge incident on either u or v

return C

This algorithm has also the approximation degree of 2 because in the worst case scenario, when none of the edges share a node we choose 2\*|E| vertices instead of |E|.

A heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. The approximation algorithms are somehow similar with heuristics. A difference between them is that heuristics could derive from theory or experimental experience, but approximation algorithms have solid theory foundation (provable solution). Also for the approximation algorithms the solution can’t be worse than a given factor and some performance is guaranteed.