Chapter G4: Approximation Algorithms

When the problem is known to be NP-complete (or worse), what you gonna do? The best one can hope for is an algorithm that is guaranteed to be close. Alternatively, one can ask for an algorithm that is close most of the time, or maybe is correct but only its average running time is fast. Or maybe only works for a subset of problems.

G4.1 $c$-Approximation Algorithms

A $c$-approximation algorithm is guaranteed to be within a factor $c$ of the exact solution.

For example, we saw earlier that there is a polynomial-time algorithm for maximum matching in a graph. But there is a simple greedy 2-approximation algorithm: start with any edge and keep on adding edges to the matching until you cannot proceed. This is called a maximal matching.

**Lemma.** Every maximal matching $M$ has size at least half the maximum matching number

**Proof.** Let $M^*$ be a maximum matching. Since $M$ is maximal, none of the edges of $M^*$ can be added to $M$. Thus for each edge of $M^*$, one of its ends must already be used in $M$. Thus the total number of nodes in $M$ must be at least the number of edges in $M^*$: but every edge has two nodes, whence the result.

The result is accurate. For example, if one has a path with three edges, then the maximum matching number is 2, but one can get a maximal matching with just the central edge.

G4.2 Vertex Cover

The vertex cover problem in a graph is to remove the minimum number of nodes such that all edges are deleted. Equivalently, a vertex cover is the complement of an independent set. Here is a 2-approximation for vertex cover.

```plaintext
GreedyVertexCover
  while there exists some edge
    take both ends of the edge and remove.
```

Note that for every edge we remove, we need at least one end in every vertex cover. So the algorithm uses at most twice what an optimal algorithm needs.
Strangely, this does not provide a good approximation to the independent set problem. Indeed, it has been shown that finding a $c$-approximation of the independence number, for any constant $c$, is NP-complete.

G4.3 Traveling Salesman Problem

Recall that a hamiltonian cycle is a cycle that visits all nodes exactly once and ends up where it started. In the traveling salesman problem TSP, every possible edge is present and each edge has a weight; the task is to find the hamiltonian cycle with minimum total weight. Since the HAMPATH problem is NP-complete, it is easy to show/believe that TSP is NP-complete too.

Now suppose the graph obeys the triangle inequality: for all nodes $a$, $b$, $c$, the weight of the edge $(a, b)$ is at most the sum of the weights of the edges $(a, c)$ and $(b, c)$. Such would be true if the salesman were on a flat surface, and the weights represented actual distances.

There is a simple 2-approximation algorithm for this version. Start with a minimum spanning tree $T$. Since a tour is a connected spanning subgraph, every tour has weight at least that of $T$. Now, if one uses every edge of $T$ twice, then one can visit every node at least once and end up where one started.

To turn this into a tour we simply skip repeated nodes. For example, in the picture the traversal of $T$ goes $ABACDFDCECA$ and we short-cut this to $ABCDFEA$. We rely on the triangle inequality for the shortcut to be no worse than the original path.

G4.4 Coloring 3-colorable graphs

A graph is defined to be $k$-colorable if one can color each node, using a palette of $k$ colors, such that no edge has both ends of the same color. A graph being bipartite is equivalent to it being 2-colorable.

It is known that testing whether a graph is 3-colorable is NP-complete. But consider the following problem: suppose you know that the graph is 3-colorable (or at least your
algorithm need only work for 3-colorable graphs). How easy is it to find a 3-coloring? Well it seems to be very difficult. For a long time the best polynomial-time algorithm known used $O(\sqrt{n})$ colors to color a 3-colorable graph. This algorithm rested on two facts:

1. One can easily 2-color a bipartite graph.
2. If a graph has maximum degree $\Delta$, one can easily produce a coloring with at most $\Delta + 1$ colors.

The only fact about 3-colorable graphs that we need is that in a 3-coloring, the neighbors of a node are colored with two colors; that is, the neighborhood of a node is bipartite. Proving this and putting this all together is left as an exercise.

G4.5 Fixed-Parameter Tractability

Finally, we consider another measure of the difficulty of a problem. A parameter $\pi$ is called fixed-parameter tractable if there is an algorithm for the problem $\pi(G) \leq k$ that runs in time $f(k)n^C$ for some constant $C$ and function $f$. We noted above that testing $k$-colorability doesn’t have this property (assuming $P \neq NP$).

Vertex cover is known to be fixed-parameter tractable. Here is the algorithm. It uses kernelization; meaning we trim the graph to a constant-size graph.

```
VertexCover(Graph G, integer k)
  1. while graph has node v of degree more than k
    take v
  2. if remaining graph has more than $k^2$ edges
    then Reject
  3. test all sets of $k$ nodes (ignoring isolated nodes)
```

In the first step, if we do not take a node of degree more than $k$, then we have to take every neighbor, which would already exceed the allowance. In the second step, if we reject it is because it would take more than $k$ nodes to cover all remaining edges. If we reach the third step, we have at most $k^2$ edges. Isolated nodes are irrelevant and so can be discarded. It follows that the number of nodes remaining is at most $2k^2$. And now, simply use exhaustive search. Yes, it’s exponential in $k$, but that doesn’t matter for fixed-parameter tractability.
Exercises

1. Give a fast algorithm to 2-color a bipartite (meaning 2-colorable) graph.

2. Give a fast algorithm to color the nodes of a graph with at most $\Delta + 1$ colors, where $\Delta$ denotes the maximum degree.

3. Hence show how to color a 3-colorable graph with $O(\sqrt{n})$ colors.

4. The $k$-center problem is to place $k$ facilities (e.g. hospitals), such that the farthest that anyone has to travel to the nearest facility is as small as possible. Use a greedy approach to produce a 2-approximation for fixed $k$.

5. Use dynamic programming to obtain an algorithm for TSP that runs in time $O(2^n)$. 