CS 170
Spring 2016

Efficient Algorithms and Intractable Problems
Alessandro Chiesa and Umesh Vazirani

Final Project

Andrew Wong, Derrick Mar, Nader Azari, Lucine Oganesian

We designed a greedy depth-first search algorithm. Our greedy algorithm was designed to have a variable 'branching factor,' which corresponded to the number of children that would be considered; 'branching factor' number of randomly selected children were chosen to be explored from each vertex. After depth-first search found a cycle, we would store that cycle and remove all vertices that were a part of that cycle, thus minimizing the number of vertices to be searched over in future iterations. Further optimizations included running multiple iterations of greedy to find a better solution when the instance finished under a given time limit.

However, greedy did not always perform well, so we devised an alternate solution. In the alternate solution, breadth-first search was used to find as many cycles of length 5 or less. To expedite the process, only 25% of the children of each vertex were randomly selected to be explored. After finding as many cycles as possible, the problem reduced to a set packing problem (see source 1), wherein we needed to figure out which sets to select to maximize our reward (i.e, the penalty we would otherwise receive if we didn't select a cycle), while also making sure that no element (vertex) was selected more than once. We formulated the problem as a relaxed integer linear program (see source 1):

$$\max_x \sum_{i=1}^{n} c_i x_i$$

$$0 \leq x_i \leq 1, \ \sum x_i \geq 0, \ \sum_{i:v \in S_i} x_i \leq 1 \ \forall v \in G$$

where x_i is 1 if cycle S_i is selected, 0 otherwise and v is a vertex in the given graph G. We used PuLP (see source 2), a python library for solving linear program, to solve our linear program. We had to limit the number of sets (i.e. cycles) considered to be at most 6000 (for computation time). These were randomly selected from all the cycles found by BFS.

For each instance we selected either the output of greedy or the output of the linear program, depending on which maximized our reward the most. The linear program failed to yield a timely answer on one of the instances (238), for which we took greedy's solution.

Sources

1. Set packing https://en.wikipedia.org/wiki/Set_packing#Integer_linear_program_formulation
2. PuLP documentation http://pythonhosted.org/PuLP/CaseStudies/a_set_partitioning_problem.html