Parallalized Solver for General Graph Problems in NP
15418 Project Proposal

Sean Zhang xiaoronz
Yucen Li (Lily) yucenl

Our project will be available at `http://www.andrew.cmu.edu/user/xiaoronz/`.

1 Summary

We plan on implementing a domain-specific language for solving NP-complete graph problems. The language will be used for graphs with edges and nodes, and the searching for the solution will be automatically optimized based on features of the machine as well as dynamic programming.

2 Background

NP-complete problems are interesting because they can be verified in polynomial time, but no known algorithms exist for finding the solution in polynomial time. Therefore, given a NP-complete problem, the only general way to determine the solution is to use the brute-force method and try every single possible configuration within the solution space. The number of possible solutions is exponential with respect to the input length, and the search takes a very long time to run.

However, many of the problems can be solved in pseudo-exponential time with the use of dynamic programming. When using the brute-force method, many of the same sub-problems are often recomputed. For example, in the knapsack problem, we are given a set of items each with a weight and value, and then asked to determine the items to include so that the total weight is less than a specified limit and the value is as large as possible. When running through all solutions, we will often recompute the weight and value of a specific subset of items. This can be optimized through dynamic programming, because the results of these computations can be stored. However, this method of solving the problem can be difficult to program correctly and often involves complex decisions relating to data structures. Many programmers may choose to brute-force the solution without optimizing based on sub-problems.

In our project, we hope to abstract away some of the detailed implementation by using a domain specific language to automatically convert an input into the corresponding dynamic programming method. Additionally, we plan on checking multiple instances of the problem in parallel, since much of the checking can be done independently.

3 Challenges

When solving instances of NP-complete problems, many of the tasks are not dependent. Checking each possible solution can be greatly optimized by running them in parallel. Additionally, some NP-complete problems compute similar sub-problems, which can be stored and accessed through a data structure similar to a matrix.

The first challenge is to find a representation for the NP-complete problems. There are a wide variety of different problems, and it does not seem immediately obvious which representation would be the best fit for our programming language.

Additionally, it may be difficult to parallelize the search for solutions across the different threads. Even though the solutions have similar computation times for verification, it may be difficult to find efficient ways for threads to communicate and keep relevant info in the cache without running into problems such as false sharing.

It may also be difficult to automate the dynamic programming. It seems that the sub-tasks which exist may be specific to a particular instance of an NP-complete problem, and it may not be easily generalizable to all programs. Therefore, even if a task could be run quickly with dynamic programming, it may be difficult for our programming language to determine the most efficient way of finding the solution.

4 Resources

We will start from scratch and implement the simple, brute-force algorithm. We will determine what type of machine we will optimize the code on.

5 Goals and Deliverables

For our project, we have a few main goals that we want to achieve:

1. Create a programming language interface which represents (possibly just a subset) of NP-complete problems
2. Parallelize the search for solutions, optimized depending on the number of threads available
3. The exact performance improvement depends on the specific problem and the machine, but in general, we expect to be able to solve larger instances of NP-complete problems.
4. If applicable, automatically set up dynamic programming so the different sub-problems do not need to be repeatedly computed across multiple threads

If the project goes really well, we can also try the following

1. We can consider parallalize randomized or approximate algorithms for graph problems.
2. We can also consider another subset of NP-complete problems, such as satisfiablity of boolean formulas, and parallalize the computation.

Additionally, some more modest goals that we can achieve are:

1. Design parallel algorithms to solve vertex cover and 3-coloring. Optimize them in a multi-thread environment.
2. Design an parallel algorithms that implements the solution to the knapsack problem with dynamic programming.

5.1 Final

For our final project, we would ideally be able to use our programming language to create a few examples NP complete problems and show how our language automatically searches for the solution.

6 Platform Choice

We plan on implementing the functionality in Cython. Due to the overhead of Python and its slow execution speed, we wanted to work in a language that was faster and more efficient. We also like how Cython has many libraries available to support the parallelizing of solutions. The objects in Cython are also beneficial.

7 Schedule

- 11/9: Implement vertex cover and optimize based on number of threads specified
- 11/16: Implement 3-coloring and optimize
- 11/20: Find implementation which is representative of all NP-complete graph problems
- 11/30: Generalize speedup to all NP complete graph problems
- 12/7: Add section allowing users to specify which sub-tasks are shared between for optimization, and automatically converting the program to use dynamic programming
- 12/15: Finalize project, polish the syntax, and work on writing project report and poster