Parallalized Solver for General Graph Problems in NP
15418 Project Checkpoint

Sean Zhang xiaoronz
Yucen Li (Lily) yucenl

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

1 Modified schedule

- 11/21: Lily: try other implementations for the 3-coloring problem, with a focus on coming up with alternative data structures that take advantage of the structures in special cases of the problem (for example, maybe we can consider a different data structure for large, sparse graphs). Sean: the same thing, but for the vertex cover problem.
- 11/25: Find common features among the data structures, generalize to our implementation of the graph interface. Lily and Sean can each implement some of the data structures for the general case.
- 11/28: Generalize the operations in our two programs to functions in our interface. Create speedup plot for other NP-complete graph problems, and consider if more speedup is possible in the general case. Lily and Sean can work on this part together.
- 12/2: Start working on the dynamic programming part. Sean can work on solving the Traveling Salesman Problem by implementing the Held–Karp algorithm. Lily can work on the Bellman-Ford algorithm for the shortest path problem.
- 12/5: Optimize the parallelization of the two problems above by considering different data structures for different sizes of problem instances. Sean and Lily can each work on optimizing their own problem.
- 12/9: Generalize to an dynamic programming interface, similar to the one for NP-complete graph problems. If we have more time, we can try automatically converting the program to use dynamic programming. Sean and Lily can work on this part together.
- 12/12: Sean can work on adding an interface that allows users to specify which sub-tasks are shared between for optimization for the graph problems, as well as other user-defined optimization possible for the dynamic programming part. Lily can work on finalizing our interface, as well as on writing the project report and creating the poster.
- 12/16: Finalize our report and poster, print out the poster before the poster session. Sean and Lily can work on this part together.

2 Work so far

For our project, we have worked on implementing specific instances of NP-complete problems, specifically 3-coloring and vertex cover.

2.1 3-coloring Problem

For the 3-coloring problem, we have developed a sequential algorithm which uses brute force to search for all possible solutions. This search was easily parallelized through a message-passing based implementation, where each thread would only compute a subset of all possible solutions. The threads each pass a message containing a boolean variable indicating if the valid solution was found. If the solution was found, then a structure containing the valid solution was also passed back to the main thread. However, due to possible synchronization hangups, the amount of speedup obtained does not seem to be very high. Therefore, it may be worthwhile to consider other data structures and implementations, as well as reorganize the code for better memory performance. Since the code is run on graphs, which have low arithmetic intensity, it may be worthwhile to consider a few of the graph optimization tricks covered in class.

2.2 Vertex Cover Problem

For the vertex cover problem, as a first step, we also developed a sequential algorithm which uses brute force to search over all possible solutions. Here we consider the decision version of the problem. We first generate all possible vertex cover with k vertices, and we check individually if each one of them is valid.

We then parallelized the search by using multiple threads, where each thread gets an equal amount of possible vertex covers, as well as an equal amount of vertices to check within each vertex cover. All vertices have access to the graph, which is currently stored as an $n \times n$ boolean matrix. The speedup due to parallelization is better than the 3-coloring case, since the matrix data structure allows us to divide work pretty evenly and also has good spacial locality. But storing the graph with n^2 space may not be ideal, especially with large but sparse graphs. To handle this case, we will implement the same algorithm but with linked lists. We expect the speedup to be worse due to the fact that cache misses are more likely and division of work will be more uneven.

3 Goals and Deliverables

Our goals and deliverables have not changed much compared to those listed in our project proposal, which were:

1. Create a programming language interface which represents NP-complete graph 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.

We are confident that we can complete (1), (2), and (3). For goal (4), since we have not started the dynamic programming part, it's a bit hard to estimate whether we can complete it or not.

Our deliverable will be two interfaces: the first one is for solving graph problems with parallelism, the second is for solving problems with dynamic programming.

Judging from our current schedule, we probably do not have time for our two extra goals that we wanted to achieve. Those two goals were:

1. Considering randomized or approximate algorithms for graph problems.
2. Consider another subset of NP-complete problems other than graphs.

For the more modest goals that we listed last time, we have partially achieved the first one: solving vertex cover and 3-coloring and parallelize the computation. We are confident that we can solve the second part: implement two dynamic programming problems and optimize them with parallelization.

4 Goal for poster session

For our poster, we plan to demonstrate what we have completed using graphs. For the graph part, we will create speedup plots for the two problems that we specifically optimized. We will also create speedup plots for other NP-complete problems solved using the interface that we created. In particular, we will also explain how problem size changes the implementation and illustrate how problem size affects the speedup. For the dynamic programming part, we plan to do the same thing, with speedup plots for the two problems, plots for other general problem using our interface, and plots for different problem size. If we completed goal (4), we will also show graphs on the speedup for the automatic dynamic programming interface compared to a dynamic programming algorithm written by humans.

5 Issues and remaining problems

For the remainder of the project, we would like to generalize our results for NP-complete problems and find a simple and composable solution. Additionally, since many of the NP-complete problems are represented as graphs, many of the operations have low arithmetic intensity. Therefore, it may be worthwhile to consider a few of the graph optimization tricks covered in class such as graph compression in order to increase the performance.