Rochester Institute of Technology
Master’s Project Proposal
Michael Adams

A Parallel Framework for NP Combinatorial Optimization Problems

Department of Computer Science
Chairperson: Alan Kaminsky
Reader: Stanislaw Radziszowski
Observer: Leon Reznik
I. Summary

For my master’s project, I plan to design, implement, and demonstrate the effectiveness of a parallel framework for NP optimization problems, using Professor Kaminsky’s Parallel Java library. This framework will be designed such that the user will only be required to implement problem specific details, and where applicable, solution strategy details. It will provide three solution strategies, each supporting parallel operation. The first strategy will be an exhaustive brute force solution. For this strategy, the possible solution space will be divided amongst the available processors, each of which will search through its section of the solution space for the optimal answer. Once each processor has exhausted its solution space, the final chosen solution of each processor can be compared, and the globally optimal answer will be known. The second strategy will be a simple hill climbing strategy. In this strategy, each of the available processors will start at a random solution in the solution space. From this solution, small adjustments will be made in an attempt to improve the solution. For example, in the knapsack problem, this could involve including an item that was previously excluded, or vice versa. These adjustments will continue until no other adjustments improve the solution – that is, once a local minimum or maximum has been reached. This strategy will also allow for some customization, such as how many of these adjustments to make at a time (e.g. how many items to include/exclude at once). Once all processors reach a local minimum or maximum, their final solutions will be compared against each other, and the best available solution will be chosen as the final solution. The last strategy will be simulated annealing. Similarly to hill climbing, each available processor will start at a random point in the solution space. Again, slight adjustments will be made, but adjustments that result in a lower quality solution can be accepted based on some cooling schedule derived probability. The user will be able to provide specific details for the cooling schedule.

To demonstrate the effectiveness of this framework, I will implement three NP optimization problems using the framework. Each of these problems will be able to use all three of the solution strategies. The problems I have chosen to implement are the knapsack problem, maximum satisfiability, and minimum vertex cover. For each problem, I will have to specify the details that correlate to that problem. For example, I will have to specify what is being included/excluded, such as a profit/weight item for the knapsack problem, or a graph vertex for the minimum vertex cover. In addition, I will have to specify what constitutes a solution, as well as how to evaluate such a solution.

Once all three problems have been implemented and tested, I will write a report covering the design, implementation, and results for the three problems.

II. Overview of Project Area

The NP problem domain encompasses problems that, using the best methods we know, increase exponentially in difficulty as the problem size increases linearly. In many problems, the only known way to find the exact optimal solution is to try every possible solution that exists. The particular area that this project will focus on is NP optimization problems, in which a certain function must be minimized or maximized. There is a great interest in the NP domain problems,
as many real life problems can be expressed in such a way that they may be solved as an NP problem. As an example, determining which hallway intersections provide the minimum number of security cameras while still monitoring all hallways can be expressed as a minimum vertex cover problem. Some solution strategies are inherently more complicated than others. Brute force may be the simplest to implement, and guarantees an optimal solution, but for larger problem sizes it can take an unacceptable amount of time to finish. Thus, heuristics such as simulated annealing can be used to find a good, but not necessarily optimal, solution in a reasonable amount of time. Many of these problems exhibit similar characteristics that can be abstracted out. Many optimization problems deal with some sort of “items”, each of which may be “included” or “excluded” as part of a solution. Each solution has a particular “value” or “validity”, which can be compared against other solutions as an indicator of quality or correctness. Since many of the solution strategies simply deal with this concept of an item, a solution, and a way of evaluating a solution, these algorithms can be written to work with abstractions of these concepts. In such a case, the solution strategies could be applied to a particular problem by specifying what an item is for that problem, what a solution constitutes, and how to evaluate a solution and compare it against others. This would significantly simplify the process of implementing a solver for a particular NP optimization problem. This framework could also prove useful outside the domain of NP, such as coNP or PSPACE. In addition, many of these solution strategies support parallel execution, which is also independent of the problem. Given the current trend of hardware advancement, parallelism will be a strong focus for advancing computer algorithms. This is where the inspiration for the proposed framework was derived.

III. Hypothesis

I hypothesize that the framework I am proposing will provide ease of use when implementing NP optimization problems in parallel. I also hypothesize that the framework’s parallelism will be beneficial to each search strategy in comparison to a sequential version. For the brute force strategy, I hypothesize that this benefit will be a speedup over a sequential execution. For the other two strategies, I hypothesize that the benefit will be a higher quality solution than what might be found sequentially.

IV. Evaluation Metrics

To evaluate my first hypothesis, I will implement a problem both in the framework as well as from scratch. For each implementation, I will record how long the implementation took me, as well as how much lines of code I had to write. This will be the metric used to compare ease of use of the framework in comparison to having to implement a problem without the framework.

To evaluate my second hypothesis, I will implement a sequential version of the framework, and run example problems using both the sequential and parallel versions. The brute force strategy hypothesis will be evaluated by looking at the amount of time required to solve the problem both sequentially and in parallel. If my hypothesis holds true, then the running time of the parallel implementation should be shorter than the sequential implementation, proportional to the number of processors available. The other two strategies will also be evaluated by example through both the sequential and parallel implementations. The metric used will be the number of
times the parallel implementation was able to find a better solution than the sequential implementation. Since the sequential version only runs on one processor, this will show that having multiple cores running individually increases the chance of finding a higher quality solution.

V. Design Specification

The proposed framework will be designed in an object oriented fashion, with a goal of maximum code reusability. There will be a concept of a solution strategy, which will further be implemented into the three proposed solution strategies (e.g. there will be something like a SolutionStategy class, which will be the superclass of the three strategies). These three strategies will further allow users to plug in details of that strategy, such as a cooling schedule for a simulated annealing solver. Another benefit of this is that, if the user wishes to, they can implement another solution strategy under this superclass. The concept of an optimization problem will also be abstracted into a superclass. This class will work on abstract concepts like a data item, a solution, and an evaluation of a solution. The specifics of what a data item is, etc., will have to be implemented by the user. Once implemented, this problem class will be able to execute the different solution strategies, and this will create a set of solutions. The user can implement how this solution will be saved or printed to standard output.

Figure 1 shows a rough example of some of these classes. The NPOptProblem class would likely be an abstract class. It would contain elements common to all NP optimization problems, such as a best solution found, and a list of data items. Again, these data items are an abstract concept, and thus would be represented as an abstract class or perhaps an interface, shown below as DataItem. Thus, the framework would be written to handle DataItem objects, and the user will specify the details of their problem’s data items. Below, this can be shown with the example of a KnapsackItem. The user would implement this class with a profit and a weight, and it could then be passed in as a DataItem. These would be used by the user-implemented KnapsackProblem class. In this class, the user would implement the “evaluate(Solution)” method. Specifically, this evaluate method would have to look at the problem’s list of data items, in combination with the solution being evaluated, to determine how good this solution is (e.g. the summed profit, and whether or not the summed weight is under the weight constraint). The Solution class would represent one possible solution in the solution space. A particular solution will contain an Assignment object. This Assignment object will in some way represent the inclusion/exclusion of the problem’s data items. While this may take many forms, an intuitive form is a binary representation of 0s and 1s, held in some particular data structure (byte, int, etc.), where 0s could represent exclusion and 1s could represent inclusion. The concept of a solution strategy is represented by the class SolutionStrategy, which will be an abstract class or an interface. The different solution strategies will then fall under this class. This is used in the NPOptProblem’s “solve(SolutionStrategy)” method. This method would use the provided solution strategy to generate a set of potential solutions. Each of these solutions will be evaluated using the user-implemented evaluate method, and the best solution found will be stored as a local variable in the problem class. For example, if the BruteForce solution strategy is passed in, the solve method will generate all possible solutions, evaluate them, and store the optimal solution. If
HillClimbing is passed in, a set of random solutions will be generated, and each of these solutions will hill climb to local minima or maxima. The number of solutions generated will be something the user specifies when instantiating a HillClimbing object. A similar pattern would be used for the SimulatedAnnealing strategy. Again, a specific CoolingSchedule can be implemented and passed in by the user.

![Diagram](image)

**Figure 1** High level design of the proposed framework

VI. Principal Deliverables

The deliverables for my project will include source code, a developer’s and user’s manual, and a report. The source code will be fully commented in javadoc style, and will be in the form of a JAR file. The developer and user manuals will include instructions for implementing a problem using the framework, compiling it, and running it. The report will cover the inspiration for the framework, as well as the process of designing and implementing it. The results of the three demonstration problems will also be found in the report.
VII. References


VIII. Schedule

February 8th  First draft of proposal completed
February 15th  First draft of class design and diagrams, proposal polished
February 22nd  Framework design completed
March 1st  Sequential framework initial implementation
March 8th  Sequential framework polished, parallel framework initial implementation
March 15th  Parallel framework polished, initial implementation of three problems using framework
March 22nd  Three problems polished, and initial implementation without framework
March 29th  Three problems polished without framework
April 5th  Additional polishing and commenting of code, buffer room
April 12th  All source code completed and commented
April 19th  All experiments (demonstration runs) completed and data collected
April 24th  First draft of report completed
May 1st  Final draft of report completed
May 10th    Target defense date