Team Amalgam

SE390 Research Plan

Joseph Hong, Chris Kleynhans, Ming-Ho Yee, Atulan Zaman
{yshong,cpkleynh,m5yee,a3zaman}@uwaterloo.ca

February 10, 2013

Abstract

Moolloy, a tool that solves multi-objective optimization problems, is greatly limited by the time it takes to solve large problems. Our work is to optimize Moolloy so it can solve these problems. We have identified case studies in three fields: aerospace, civil engineering, and software engineering.

Contents

1 Problem Definition **2**

2 Related Work **3**

3 Research Value **3**
3.1 Aerospace 3
3.2 Civil Engineering 4
3.3 Software Engineering 4

4 Goals **4**

5 Methodologies **4**

6 Risks & Technical Feasibility **6**

7 Costs **6**

8 Legal, Social, and Ethical Issues **7**

References **7**

1 Problem Definition

Multi-objective optimization is a widely researched area of computer science that focuses on finding solutions to problem definitions with respect to given objective realization constraints. Computing such solutions is extremely resource intensive, and the computation time grows exponentially with the number of optimization variables.

The nature of our work in scientific terms is called *exact, discrete multi-objective optimization.* Multi-objective optimization (MOO) is the process of computing the most optimal solutions given a goal and a set of constraints.

Multi-objective means that there are multiple metrics that must be optimized over. Thus, more than one optimal solution may be found that satisfies the various optimization goals.

Exact in the context of our problem indicates that all solutions computed by the algorithm are Pareto-optimal. Informally, this means none of the solutions can be improved without compromising at least one other optimization goal. Heuristic approaches for multi-objective optimization do not guarantee that all their solutions are Pareto-optimal.

Discrete indicates that our optimization algorithm only addresses discrete, or combinatorial, input data. The algorithm does not accept continuous optimality conditions.

One simple example of a multi-objective optimization problem is the satellite scheduling problem. In this problem [1], NASA must determine the best possible satellite launch schedule that maximizes scientific value. Each satellite has a different cost, purpose, and value to a different scientific community. In this problem, the constraints for NASA are such things as resource limitations and launch ordering constraints.

Moolloy [2] is a tool introduced by the MIT Computer Science and Artificial Intelligence Laboratories that solves multi-objective optimization problems. It also has a GUI that lets the user specify the constraints and objective conditions, as well graphically view the Pareto-optimal solutions computed by the algorithm. The underlying algorithm, called the *guided improvement algorithm*, is capable of solving small problems. However, its scalability is greatly limited by the time it takes to compute solutions for large problems, since the algorithm must make multiple calls to a SAT solver.

Therefore, our work will focus on increasing the performance of Moolloy, while ensuring the algorithm continues to perform correctly.

2 Related Work

The *guided improvement algorithm* was described by Rayside, Estler, and Jackson [2] in 2009. In their paper, they also conducted a literature survey on related work. Rayside et al. found that most multi-objective problems were concerned with continuous variables, as opposed to discrete variables.

Furthermore, most of the research on multi-objective optimization they found was focused on heuristic approaches, specific instead of general solvers, extensions of single-objective solvers, or problems with only two or three variables.

The guided improvement algorithm is an exact, discrete, general-purpose solver. Furthermore, it is not an extension of a single-objective approach. Rayside et al. identified a similar approach they call the *opportunistic improvement algorithm*, which was independently discovered by Gavanelli [3] and Lukasiewycz, Glaß, Haubelt, and Teich [4]. Notably, the guided improvement algorithm produces intermediate Pareto-optimal results during its computation.

Only three publications have cited the guided improvement algorithm paper since it was published. One of them describes a spreadsheet-like user interface, while the other two concern applications of the algorithm. Thus, none of them are related to our work, which is strictly to optimize the algorithm.

A more recent paper by Dhaenens, Lemesre, and Talbi [5] proposes *K-PPM*, an algorithm which can be parallelized. However, the algorithm does not produce intermediate Pareto-optimal results.

3 Research Value

Multi-objective optimization problems appear in may different fields. By improving the performance and scalability of the algorithm, we will enable its usage for problems with larger input spaces. We have identified potential case studies from three different fields: aerospace, civil engineering, and software engineering.

3.1 Aerospace

Every ten years NASA performs its decadal survey to determine the satellite launch schedule for the next decade [1]. Multi-objective optimization can be used to determine a schedule that maximizes scientific value to different scientific communities while minimizing cost. Furthermore, many different

constraints must be satisfied. For example, one satellite may depend on another, or a satellite may need to match a specific timeline.

3.2 Civil Engineering

Dr. Bryan Tolson, a civil engineering professor from the University of Waterloo, has identified a number of multi-objective optimization problems in his research. Currently, the problems are solved using heuristic methods, in other words, genetic algorithms. As discussed earlier, these methods do not guarantee that their solutions are Pareto-optimal. One of Dr. Tolson's problems is to determine the optimal type, quantity, and arrangement of materials to line a landfill in order to minimize seepage, while keeping cost minimal.

3.3 Software Engineering

We will be collaborating with Rafael Olaechea, a graduate student at the University of Waterloo who is interested in the *software product lines* problem [6]. This problem involves determining which modules should be included in the software for an embedded device. Each module can perform different functions, which may conflict with other modules. Additionally, each module will have a different cost in terms of code size and performance metrics. Therefore, the problem is to determine an optimal set of modules for a device, while satisfying the functionality constraints.

4 Goals

The goal of this project is to optimize Moolloy by reducing the number of SAT solver calls. This will increase the scalability of Moolloy so it can successfully compute solutions for large multi-objective optimization problems that are not possible in the current version. To measure success, we will create a benchmark suite to confirm that our versions actually make Moolloy faster. Furthermore, we will develop a regression suite to ensure Moolloy continues to compute correct solutions.

5 Methodologies

Before we begin the optimization work, we will identify a series of problems that can be used as test data. These problems will range from small benchmarks to test correctness, as well as large, real-world problems to test

for optimization. We will also begin converting these problems into models that Moolloy can take as input. Concurrently, we will perform preliminary profiling on Moolloy to identify potential bottlenecks. Next, we will begin refactoring the Moolloy code base, to make it more maintainable.

Once these tasks have been completed, we can begin exploring different optimization strategies, which are listed below:

Parallel Decomposition
: Determining a way to split the search space into pieces each of which can be searched independently in parallel.

Input Space Reduction
: Determining a way to prune configurations from the search space. For example, if we know that any optimal configuration must have a certain property, we can eliminate configurations without that property.

Empirical Profiling
: Determining empirically which types of SAT solver calls are most expensive and reducing these types of SAT calls. Determining if the bottleneck is indeed the SAT solver and optimizing other components if not.

Improve Search Guidance / Speculative Execution
: Instead of trying to find a solution that is better, we place constraints on how much better it must be (for example, twice as good) which may reduce solver time.

Workflow Feedback
: Often a user will start the program running and then realize from partial results that he or she made a mistake. Can these partial results be saved and used to speed up the next run?

Incremental SAT Solving
: Currently we start an entirely new SAT solve for each search, can we save some of the knowledge gained from initial solves to speed up later searches?

Genetic Algorithm Seeding
: Use a genetic algorithm solving method to get approximate Pareto points, and then use these approximate points as the starting points for the guided improvement algorithm.

Overlapping Parallel Searches

Find N different solutions, where N is some function of M, the number of metrics. From the N seed points, start parallel searches for the Pareto front. Eliminate any duplicate Pareto points.

These ideas are the result of previous work done by Dr. Rayside and his collaborators [7]. As we proceed with the project, we may eliminate some of these optimization ideas, or add our own.

6 Risks & Technical Feasibility

One risk in this project is that we may not be able to find good test cases to evaluate our optimization techniques. Each problem is different, and thus there are many possible routes for optimization. Furthermore, significant domain knowledge may be required to properly model the problems. This is why we are working with multiple collaborators in different fields, to obtain both test problems and relevant domain knowledge.

An unlikely risk for this project is that none of our techniques provide satisfactory results in terms of performance. There is still research value, although very little research impact, as there is very little benefit to users.

7 Costs

Based on informal calculations, we estimate that it would cost $50,000 to pay the four developers on this project. This estimate is based on the average co-op student hourly earnings information [8] provided by the University of Waterloo. We assume an approximate total of 500–600 developer-hours, spread over three study terms and two co-op terms, to complete the project.

Aside from the cost associated with the labour, we estimate our expense to be around $350 as a result of purchasing various tools and services, as well as hardware infrastructure for our tests. Table 1 shows the total cost breakdown for our project.

Table 1: Total cost breakdown

Item	Cost
Developer time	$50,000
Atlassian Bamboo (continuous integration and build system) [9]	$150
Atlassian HipChat (asynchronous communication platform) [10]	$150
Amazon EC2 Instances (cloud computing infrastructure) [11]	$50
Total	$50,350

8 Legal, Social, and Ethical Issues

As we will be using many external libraries, we will need to consider which software licenses they are licensed under. Alloy and Kodkod are both licensed under the MIT License, a permissive free software license. Thus, no restrictions or limitations are applied to our work.

We may also face various legal issues with incorrect or suboptimal solutions generated by our solver. The use of such solutions without any verification may result in undesired consequences, which include, but are not limited to, bodily harm, violation of regulations, and monetary damages. As such, a disclaimer must be presented to our users, advising them that verifications be done on all solutions, and that we hold no liability in case of such circumstances.

Moreover, by increasing the optimality of our solver, we may be able to solve multi-objective optimization problems that were previously unsolvable by computers in reasonable time. This will likely cause several ethical and social issues, the most predominant issue being the elimination of job positions that were previously tasked to solve those problems manually.

References

[1] NASA. (2011). Decadal Survey, [Online]. Available: `http://science.nasa.gov/earth-science/decadal-surveys/` (visited on 11/25/2012).

[2] D. Rayside, H.-C. Estler, and D. Jackson, "The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization," MIT Computer Science and Artificial Intelligence Laboratory, Tech. Rep. MIT-CSAIL-TR-2009-033, Jul. 2009.

[3] M. Gavanelli, "An Algorithm for Multi-Criteria Optimization in CSPs," in *Proceedings of the 15th European Conference on Artificial Intelligence*, F. van Harmelen, Ed., Lyon, France: IOS Press, 2002, pp. 136–140.

[4] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Solving multi-objective pseudo-boolean problems," in *Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, ser. SAT'07, Lisbon, Portugal: Springer-Verlag, 2007, pp. 56–69.

[5] C. Dhaenens, J. Lemesre, and E.-G. Talbi, "K-PPM: A new exact method to solve multi-objective combinatorial optimization problems," *European Journal of Operational Research*, vol. 200, no. 1, pp. 45–53, 2010.

[6] R. Olaechea, S. Stewart, K. Czarnecki, and D. Rayside, "Modeling and Multi-Objective Optimization of Quality Attributes in Variability-Rich Software," in *International Workshop on Non-functional System Properties in Domain Specific Modeling Languages (NFPinDSML'12)*, Innsbruck, Austria, October 2012.

[7] M. Kovacevic, D. Rayside, and S. Sun, "Moolloy Algorithm Ideas," 2011.

[8] Co-operative Education & Career Action, University of Waterloo. (2011). Hourly earnings information (Jan.-Dec. 2011), [Online]. Available: `https://uwaterloo.ca/co-operative-education/why-co-op/co-op-earnings/hourly-earnings-information-jan-dec-2011` (visited on 11/28/2012).

[9] Atlassian. (2011). Atlassian Bamboo, [Online]. Available: `http://www.atlassian.com/software/bamboo/overview` (visited on 12/12/2012).

[10] ——, (2012). Atlassian HipChat, [Online]. Available: `https://www.hipchat.com` (visited on 12/12/2012).

[11] Amazon. (2012). Amazon Elastic Compute Cloud (Amazon EC2), [Online]. Available: `http://aws.amazon.com/ec2/` (visited on 12/12/2012).