Exact, Discrete, Multi-objective Optimization

ECE498A Project Specification

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

June 6, 2013

Contents

1 Motivation

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 Intended Users & 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 who shall directly benefit from the work of this research: 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.

3.4 Potential Users

Besides the specific use cases mentioned above, the concept of multi-objective optimization is useful in many other fields of engineering and with the success is scalability it will be applicable to many other domains, eg. the domain of scheduling problems similar to the aerospace problem which can be extended to more domains.

4 Overall Description

The overall description of the system is introduced by first declaring some of the technical definitions, acronyms and abbreviations that will be used throughout the document and the feature descriptions. It also introduces some of the technical terms used in the write-up for users not familiar in this domain. Followed by the definitions, a system block diagram is presented

to illustrate the interaction between the different elements in the system. A table of description is also provided for each node in the block diagram.

4.1 Definitions, acronyms, abbreviations

Definition 1. A solution is said to be *Pareto-optimal* if and only if it is not *dominated* by any other solution. A solution a dominates a solution b if all metrics of a are greater than or equal to their corresponding metrics of b, and there exists some metric of a that is strictly greater than its corresponding metric of b.

Definition 2. The set of all Pareto-optimal solutions is called the *Pareto front*.

Definition 3. A *multi-objective optimization (MOO) problem* is a problem with multiple constraints, as well as multiple goals to optimize over.

Definition 4. An *exact* solution to a multi-objective optimization problem is the Pareto front.

Definition 5. *Discrete* in this document means that there is a countable number of configurations for every problem. This is in contrast to the continuous case. A synonym for discrete is *combinatorial*, but we will only use the former term.

Definition 6. By *general-purpose*, we mean that Moolloy can solve any multi-objective optimization problem, as opposed to a specific one.

Definition 7. *SAT*, or *boolean satisfiability*, is a problem that asks whether a given Boolean formula can be assigned values such that its evaluation is true. In other words, it asks if a given Boolean formula can be satisfied.

4.2 System Block Diagram

All multi-objective optimization problem models are first converted to relational models using the Alloy modelling language. As input, Moolloy takes a multi-objective optimization problem modelled in Alloy, and returns the Pareto front as an Alloy instance. Moolloy, as an extension of Alloy, is written in Java. Moolloy compiles the alloy models and compiles into an intermediate form which is passed to the Kodkod solver. Kodkod in turn takes the intermediate forms and compiles them into SAT formulaes which can be used to makes calls to an external SAT solver. The satisfiable instance produced by the solver is then returned to Kodkod, and Kodkod

then reverse compile it to Alloy and solution is shown as an Alloy instance. Kodkod and the SAT solver are packaged with Moolloy. Figure 1 depicts the overall structure of Moolloy.

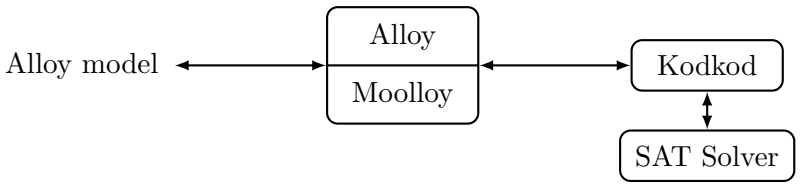

Figure 1: Overview of Moolloy structure.

We assume that all users are already familiar with multi-objective optimization, including such terms as *Pareto-optimal* and *Pareto front*. Furthermore, we assume users are familiar with how SAT solvers are used to find these solutions. The user should also be familiar with expressing the problem in Moolloy's domain specific language, as well as interpreting the results.

Some technical details behind the components of the Moolloy system is provided in the table below.

Table 1: Component Details

Item	**Purpose**	**API**
Alloy	Relational Modelling Language	Java, MIT License
Moolloy	Alloy with multi-objective solver features	Java, MIT License
Kodkod	Compiles Alloy models to SAT formulaes to call SAT solver	Java, MIT License
SAT Solver	Accepts SAT formulaes from Kodkod, solves and reports solution	Many

5 Functional Requirements

For this project, the functional requirements describe the research objectives of optimizing Moolloy. The criteria for optimizing Moolloy basically comes in two dimensions: Scalability and Speed. The basis for setting a benchmark in these criteria are explained below. These two are related, as significant improvements in speed will allow Moollloy to scale up and handle larger

inputs.

5.1 Speed

Moolloy's algorithm works by making multiple calls to a SAT solver for each multi-objective problem it must solve. These calls can be classified as SAT and UNSAT calls, where SAT calls can be satisfied, and UNSAT calls cannot be satisfied. UNSAT calls are very expensive, while SAT calls are less expensive, since they terminate once a solution is found.

Our goal is to improve the speed at which Moolloy solves a problem. This allows users to solve our current problems much faster. To accomplish this goal, we aim to reduce the number of SAT and UNSAT calls.

From Rayside et al.'s paper [2], the twenty-five variable, four metric knapsack problem takes approximately forty-five hours to complete, in the process requiring 1,938 calls to the SAT solver, where 814 were UNSAT. Our goal is to reduce this time to under twenty-four hours.

5.2 Scalability

Because solving large problems is very expensive, there are many real-world problems that simply cannot be solved at this time. We aim to optimize Moolloy so that such problems are solvable.

6 Non-Functional Requirement

These non-functional requirement are basically constraints that must considered during the development of the software project for proper software design practices and quality but are not part of the feature specifications. These criteria avoid regression of the existing Moolloy system and support design compatibility.

6.1 Design Constraints

As Moolloy is currently implemented in Java, the new version must also be implemented in Java. Furthermore, to maximize the utility of this program, Moolloy must be able to build and run on any platform that supports Java.

6.2 Preventing Regresssion

We shall be working on a repository branch of the source code for Alloy and Kodkod which shall needs to prevent regression of existing features for it to

be merged to the existing codebase therefore it is important to have proper use cases and continuous integration system while we work on the project.

7 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.

8 Costs

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 2 shows the total cost breakdown for our project.

Table 2: Total cost breakdown

Item	**Cost**
Atlassian Bamboo (continuous integration and build system)	$150
Atlassian HipChat (asynchronous communication platform)	$150
Amazon EC2 Instances (cloud computing infrastructure)	$50
Total	$350

A Guided Improvement Algorithm

This appendix outlines, at a high-level, how the *guided improvement algorithm* works. Rayside et al. [2] provide a more formal and detailed description in their paper.

First, Moolloy finds a solution that satisfies the constraints. Once this solution is found, Moolloy finds a new solution that dominates the previous one. Moolloy continues this process of finding new solutions until no other dominating solution can be found—this last solution is on the Pareto front, by definition.

Moolloy continues this process with new starting solutions (that are not dominated by any previously discovered solution) until all solutions on the Pareto front are found.

Thus, Moolloy makes a large number of calls to the underlying SAT solver. These are extremely expensive calls, so the goal for optimizing Moolloy is to reduce the number of calls to the SAT solver.

References

[1] NASA, "Decadal Survey," 2011.

[2] D. Rayside, H.-C. Estler, and D. Jackson, "The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization," Tech. Rep. MIT-CSAIL-TR-2009-033, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 7 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.), pp. 136–140, IOS Press, 2002.

[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*, SAT'07, pp. 56–69, Springer-Verlag, 2007.

[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), 10 2012.