Team Amalgam

SE390 Test Plan

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

December 2, 2012

Abstract

SE390 Test Plan for Team Amalgam

Contents

1 Introduction

Moolloy is a tool that solves multi-objective optimization problems. It is an implementation of the *guided improvement algorithm*, described by Rayside, Estler, and Jackson [1]. Because Moolloy is greatly limited by the time it takes to solve large problems, our project is to optimize Moolloy. This document serves as the test plan for our work.

The rest of this document will cover our test strategy, in particular, what tools we are using and which criteria we are testing on. We also describe four case studies.

2 Testing Strategy

The test problems will be written as Alloy models, which is the format that Moolloy expects its problems to be in. Reference solutions are Alloy instances, expressed as XML.

Where applicable, test problems will be written in Clafer [2], and we will use ClaferMoo [3] to solve them. ClaferMoo is a frontend to Moolloy, where problems are expressed in Clafer and then compiled down to Alloy. Clafer is most suitable for the software product lines problem, but may be adaptable for our other test cases.

2.1 Correctness

Our correctness tests will comprise of two distinct phases: unit testing of the individual components, and integration testing of the overall program. Unit testing will be done with the TestNG framework [4], with mocking facilitated by Mockito [5]. Integration testing consists of running Moolloy on small problem instances, and comparing the outputs to reference solutions.

These problems will be small instances of our benchmarks and case studies. This is to ensure that tests can complete quickly, providing us with quick feedback during development cycles.

Furthermore, we will utilize continuous integration during development, to ensure tests are run after every commit to the code repository. This makes it easier to identify which commit introduced buggy code. We plan to use Atlassian's Bamboo service for this purpose.

2.2 Speed

To test for speed, we will use test cases from larger instances of our benchmarks and case studies, and time how long it takes to compute solutions. However, these tests should be small enough such that the current version of Moolloy can still solve them, even if it takes forty-eight hours. Nevertheless, we intend on choosing benchmarks that can be completed in under twenty-four hours, allowing us to run nightly speed tests.

2.3 Scalability

These test cases will be very large. They cannot be solved by the current Moolloy version in any reasonable amount of time. These tests will be composed of full instances of our case study problems, as well as very large benchmarks.

As these tests are very expensive, it is impractical to run them on a regular basis. These tests will only be run on demand when we are satisfied with the results from our speed tests.

Our ultimate goal is to be able to compute solutions for these tests in under forty-eight hours. This will validate our work and show that we have met our goals.

3 Benchmarks

3.1 *n*-Queens

The n-Queens problem is to determine an arrangement of n queens on an $n \times n$ chess board, such that no two queens can attack each other. To transform this into a multi-objective optimization problem, for each metric, each square is assigned a random integer from 0 to n. To calculate the metric values for a configuration, simply sum all the corresponding values for the squares the queens are placed on. Thus, the goal is to maximize (or minimize) the metric values.

3.2 *n*-Rooks

The n-Rooks problem is identical to the n-Queens problem, except that we use rooks instead of queens. This makes the problem easier, as rooks are more restricted than queens are in their movement.

3.3 Multi-objective Knapsack

In the standard single-objective knapsack problem, a set of n items is given, where each item has a *value* and a *weight*. The goal is to maximize the sum of the values, while ensuring the sum of the weights is less than or equal some constant. As a multi-objective problem, we add multiple values and weights to each item.

4 Case Studies

4.1 Teaching Assistant Assignment Problem

In the *Teaching Assistant Assignment Problem*, a university department must assign its graduate students as Teaching Assistant (TA) positions to courses. Professors rank graduate students in order of preference, while graduate students rank the courses in order of preference. Thus, the objectives are to maximize professor happiness, graduate student happiness, and the total number of matchings.

4.2 NASA Decadal Survey

Every ten years, NASA meets with various scientific communities in order to conduct its *Decadal Survey* [6]. Each community has a number of satellites to launch, where each satellite has its own cost and value. The problem is to determine a satellite launch schedule that minimizes cost, maximizes value to the scientific communities, and respects scheduling constraints. For example, some satellites must be launched after others, or some satellites may not be ready until a certain date.

4.3 Civil Engineering Problems

We will be collaborating with Professor Bryan Tolson, from the Department of Civil and Environmental Engineering at the University of Waterloo. Professor Tolson has several interesting multi-objective optimization problems; in this document, we describe two of them.

4.3.1 Pipe network problem

In this problem, the task is to determine a set of pipes to be used for building a pipe network. Each pipe has a discrete size, and the objectives are to maximize performance and minimize cost.

4.3.2 Landfill problem

In this problem, the task is to line a landfill with various materials, minimizing seepage and cost. There are a variety of materials that can be chosen, and each material can be used in different quantities. Furthermore, the arrangement of materials will affect performance.

4.4 Software Product Lines

We will be collaborating with Rafael Olaechea, a graduate student at the University of Waterloo, on the *software product lines* problem [3]. In this problem, the task is to determine which modules of an embedded software system should be included. Performance and functionality of the modules should be maximized, while cost should be minimized. Furthermore, different modules may conflict or overlap with others.

5 Future Tests

References

[1] 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.

[2] K. Bąk, K. Czarnecki, and W. Andrzej, "Feature and meta-models in clafer: mixed, specialized, and coupled," in *3rd International Conference on Software Language Engineering*, Eindhoven, The Netherlands, October 2010.

[3] 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.

[4] C. Beust. (2012). TestNG, [Online]. Available: `http://testng.org/doc/index.html` (visited on 12/02/2012).

[5] S. Faber. (2012). mockito — simpler & better mocking, [Online]. Available: `http://code.google.com/p/mockito/` (visited on 12/02/2012).

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