Moolloy Algorithm Ideas

Michael Kovacevic, Derek Rayside, Shiqi Sun

March 18, 2011

Abstract

A discussion of ideas on how to improve Moolloy's computational performance.

Contents

1 Parallel Decomposition

How to carve up the search space into regions that can be searched in parallel. Christian Estler has some ideas about this. Beware: Ideas that seem to intuitively work in two dimensions do not always generalize to three or more dimensions.0

2 Sequential Decomposition

Consider the decadel survey problem. Is it the case that computing a schedule for the first three years produces a schedule that is a prefix of a ten-year Pareto-optimal schedule?

3 Input Space Reduction

Consider the decadel survey problem. There are some satellites that will never be launched: they are expensive and they don't offer enough benefit. At any given point in time there will always be another satellite that offers better return on investment. Eliminating these satellites from the search a priori improves search time. How to formalize and generalize this? See Anguel Nikolov's report.

Back-of-the-envelope arithmetic reveals that satellite X may have a poor cost/benefit ratio.

- Find a schedule that includes satellite X.
- Find a better schedule that excludes X.
- Find a better schedule that includes X.
- Find a better schedule that excludes X.
- ...
- If search terminates with a solution that does not include X, then X will never be launched [is this correct?]
- If search terminates with a solution that does include X, then that is a Pareto-optimal point of the original problem [is this correct?]

Assumes independence of X.

4 Duality

Are our problems convex? If so, then the duality approach may work in a relatively straightforward manner.

Assuming that we are maximizing all metrics, then the Pareto front will have a negative slope (i.e., improvement in one metric results in a decrease in another metric).

5 Empirical Profiling

Invoking the SAT solver does not always take the same amount of time. Which invocations are more expensive? Pareto-optimal points? Sub-optimal points? Some other classification of points? Empirically measure these kinds of things to see where the computational time is being spent.

How many sub-optimal points do we examine before we get to a Pareto-optimal point? Can we reduce the number of sub-optimal points examined?

etc.

6 Improve Search Guidance / Speculative Execution

Assume multiple cores.

Suppose we find our first feasible point. Right now we just search for something better. We could simultaneously search for (a) something better, and (b) something twice as good. Note that search (a) may still produce a better result than search (b): how often does this happen in practice? Should we kill one of the searches?

Update Mar 18/2011 - Mike and Shiqi - What if we had a scaling factor based on the size of problem? For example, at the beginning of the search we have a scaling of 2x and this factor continues reduction based on the number of solutions previously found. Also, what if we scaled this factor based on the ratio between the worst and best solution. For example, if we iterate through and find 5 solutions that are at least twice as good as the original solution considered, then the solution we are left with is 32x the original solution (exponential). We could scale this factor based upon this ratio.

7 Workflow Feedback

Commonly the user starts search 1, sees the results, notices a flaw in their model, revises the model, and starts search 2. Can we use the (partial) results from search 1 to speed up search 2? For example, use the points found in search 1 as seed points for search 2.