Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-033 July 3, 2009

The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization

Derek Rayside, H.-Christian Estler, and Daniel Jackson

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization

Derek Rayside[1], H.-Christian Estler[2], Daniel Jackson[1]

{drayside, estler, dnj}@csail.mit.edu
[1]MIT; [2]University of Paderborn

Abstract. This paper presents a new general-purpose algorithm for exact solving of combinatorial many-objective optimization problems. We call this new algorithm the *guided improvement algorithm* . The algorithm is implemented on top of the non-optimizing relational constraint solver Kodkod [24].

We compare the performance of this new algorithm against two algorithms from the literature (Gavanelli [11], Lukasiewycz et al. [18], Laumanns et al. [17]) on three micro-benchmark problems (n-Queens, n-Rooks, and knapsack) and on two aerospace case studies. Results indicate that the new algorithm is better for the kinds of many-objective problems that our aerospace collaborators are interested in solving.

The new algorithm returns Pareto-optimal solutions as it computes.

1 Introduction

In a single-objective optimization problem, a set of decision variables are assigned values from a given domain; a solution is an assignment for which the specified constraints hold. The optimal solution is the solution with the best value, computed by applying an objective function to the assignment.

A *multi-objective optimization problem* (MOOP) is an optimization problem with several, oftentimes conflicting, objectives. In the design of a bicycle, for example, cost and performance conflict. The decision variable *frame material* might take one of the values *Aluminum* (for high performance but high cost) or *Steel* (for lower performance but lower cost).

In most cases, a MOOP does not have a single optimal solution, but a set of optimal solutions. For the bicycle problem, a range of solutions that balance cost and performance in different ways might be obtained. These solutions are optimal in the sense that the value of one objective can be raised only by lowering the value of another. Furthermore, no solution in this set is *dominated* by any other solution in the set, meaning that for each pair of solutions $\langle s_1, s_2\rangle$, s_1 has at least one objective-value that is better than s_2's corresponding objective-value. Named after the economist Vilfredo Pareto, the set of non-dominated solutions is often called the *Pareto front*.

Since MOOPs are relevant in many areas of engineering, product design and business planning, much research has been directed at solution methods. The large number of possible solutions makes it hard to find the optimal ones; in the worst case, every possible solution must be examined.

The primary contribution of this paper is the *guided improvement algorithm* (GIA): a new, exact, general-purpose algorithm for finding the Pareto-front of combinatorial multi-objective problems. This algorithm can solve *many-objective* problems: *i.e.*, problems with more than three objectives. This algorithm returns Pareto-optimal solutions as it computes, so the user does need to wait for the algorithm to terminate before having some useful output. (The adaptive ϵ-constraint method of Laumanns et al. [17] also has this property.) This algorithm works by using a non-optimizing constraint solver – in this case, Kodkod [24]. (Although most multi-objective algorithms use an underlying single-objective solver, the work of Gavanelli [11] and Lukasiewycz et al. [18] also use non-optimizing base-solvers.)

Our tool as well as some of the problems used in the evaluation are available at http://sdg.csail.mit.edu/moolloy/.

2 Related Work

There are thousands of papers on multi-objective optimization, but only a handful that are directly related to our work. In this section we situate our work within the broader literature and discuss the directly related papers.

Multi-objective optimization (MOO) may be considered as part of the more general field of *multiple criteria decision analysis* (MCDA) or *multiple criteria decision making* (MCDM) [9]. Multi-objective optimization is concerned with the computational question of finding solutions to a multi-objective problem. Other areas of MCDA/MCDM are concerned with matters such as preference elicitation [9, 26]. Indeed, only four of the twenty-four chapters in the massive collection of MCDA surveys edited by Figueira et al. [9] are concerned with multi-objective optimization, and these only with the continuous case.

Multi-objective optimization may be divided according to the kinds of decisions it considers. The continuous case is the most common. In recent years there has been increased interest in the discrete, or combinatorial, case – which is what we address here. Surveys of the literature on *multi-objective combinatorial optimization* (MOCO) include those by Ehrgott and Gandibleux [4–6, 10] and by Ulungu and Teghem [25]. Additionally, the Annals of Operations Research recently ran a special issue on the subject [8].

These surveys [4–6, 10, 26] all come to the same conclusions about the research trends in MOCO: heuristic methods; solvers for specific problems rather than general solvers; extending single-objective optimization techniques; bi-objective or tri-objective problems rather than many-objective problems (the term *many-objective* is used for problems with more than three objectives).

Our research goes against all of these trends: we have developed an exact general-purpose solver capable of many-objective problems that is not an extension of a single-objective technique. All of these other approaches are worthwhile, they just are not directly related to our approach. We are motivated in these directions through our collaboration with a group of aerospace engineers who want exact answers to their many-objective, domain-specific problems (discussed in more detail below).

The works of Gavanelli [11] and Lukasiewycz et al. [18] are the most similar to ours in that they are exact general-purpose solvers suitable for the many-objective case and are not extensions of single-objective techniques. Gavanelli [11] and Lukasiewycz et al. [18] appear to have independently discovered the same algorithm, which we refer to as the *opportunistic improvement algorithm* (OIA) below. We have implemented the opportunistic improvement algorithm within the Kodkod framework using all of the same subroutines used for our guided improvement algorithm. A comparison of these algorithms is below. An important usability difference is that our guided improvement algorithm provides Pareto-optimal intermediate results, *i.e.* all solutions yielded by the algorithm are guaranteed to be Pareto-optimal even if the user terminates the execution prematurely.

A few other researchers have made exact, general-purpose MOCO algorithms by extending single-objective optimizing solvers, such as the outer-branching technique of Junker [15] the adaptive ϵ-constraint method of Laumanns et al. [17]; and the improved ϵ-constraint method of Ehrgott and Ruzika [7].

A common way to characterize the running time of these algorithms is in terms of how many times they invoke their base-solver. Let s name the total number of logically feasible solutions, p name the number of points on the Pareto-front, and let o name the number of objectives. The outer-branching technique of Junker [15] is $O(p \times o)$, while the adaptive ϵ-constraint method of Laumanns et al. [17] is $O(p^{o-1})$. Ehrgott and Ruzika [7] do not characterize their improved ϵ-constraint method in this way.

The opportunistic improvement algorithm ([11, 18]) is $O(s)$ in the worst case. The idea, however, is that in practice the base-solver will be invoked fewer times than this for most inputs. Such is the nature of working with a non-optimizing base-solver such as a SAT solver. SAT solvers are designed to tackle problems that are, in theory, NP-complete. It turns out that, in practice, solutions can often be found in less than exponential time. In the best possible case the opportunistic improvement algorithm would invoke the base-solver exactly $p+1$ times. Reality is usually somewhere in the middle and must be measured empirically.

Our guided improvement algorithm, presented below, invokes its base-solver $O(s+p)$ times in the worst case. In the best case it invokes the base-solver $2p+1$ times. However, as with the opportunistic improvement algorithm, reality is usually somewhere in the middle and must be measured empirically.

These comparisons in terms of the number of invocations of the base-solver make two assumptions: (1) that each invocation of the base-solver takes the same amount of time, and (2) that different base-solvers take the same amount of time. While neither of these assumptions hold in general, our empirical observation is that the algorithm that invokes its base-solver the fewest number of times is usually – but not always – the fastest. Nevertheless, complexity in terms of invocations of the base solver is a useful and insightful comparison technique.

In this paper we compare our implementations of the guided improvement algorithm and opportunistic improvement algorithm with the Laumanns et al. [17] implementation of the adaptive ϵ-constraint method on three micro-benchmark and two case-study problems. This is the first paper that we are aware of that empirically compares exact general-purpose MOCO algorithms by different authors. We also examine more challenge problems than any other paper that we are aware of in this domain.

However, we were not able to run every algorithm on every challenge problem because the tools are based on different formalisms and read different file formats. For example, our tool uses a relational logic with transitive closure, which is not supported by any of the other tools. There is a need for the MOCO community to work together to create common formalisms, file formats, and challenge problems, and to organize an annual competition, as is done in the CSP, SAT, SMT, and other communities. As the first paper in this area to compare algorithms by different authors, this paper takes some preliminary steps in this direction. However, the primary purpose of this paper is to show the novelty and utility of our guided improvement algorithm.

3 Problem Statement

3.1 Problem Input

In a multi-objective optimization problem, a vector of *decision variables* $\boldsymbol{X} = [x_1, \ldots, x_z]$ is assigned a vector of *values*, called an *assignment*. Each value is drawn from a given domain, thus we sometimes refer to it as *domain value*. An assignment is *feasible* if it respects all the constraints represented by a vector $\boldsymbol{C} = [c_1(\boldsymbol{X}), \ldots, c_p(\boldsymbol{X})]$. A *feasible assignment* is also called a *solution*. A vector of metric (or objective) functions $\boldsymbol{M} = [m_1, \ldots, m_q]$ is applied to a solution to obtain a *point* (or metric values or objective values) $[m_1(\boldsymbol{X}), \ldots, m_q(\boldsymbol{X})]$.

3.2 Pareto dominance and Pareto optimality

With no lack of generality, we consider only maximization problems as every minimization can be transformed into a maximization. Two solutions can be compared based on their metric values. We make the following distinctions:

Definition 1. *Let $\dot{a}$ and $\hat{a}$ be solutions, q the number of metric functions, and let u, v be metric function indices in $\{1, \ldots, q\}$. We say*

- $\hat{a}$ (Pareto) dominates $\dot{a}$ *with respect to the metric* $\boldsymbol{M}$*:*
  $\mathit{dominates}(\hat{a}, \dot{a}, \boldsymbol{M}) \Leftrightarrow \forall u : m_u(\hat{a}) \geq m_u(\dot{a})$ *and* $\exists v : m_v(\hat{a}) > m_v(\dot{a})$
- $\hat{a}$ (Pareto) equals $\dot{a}$*:* $\quad \mathit{equals}(\hat{a}, \dot{a}, \boldsymbol{M}) \Leftrightarrow \forall u : m_u(\hat{a}) = m_u(\dot{a})$

Given a set of solutions A, we are interested in finding *maximal* or *optimal* solutions. Optimality is defined in terms of metric values:

Definition 2. *Let $\dot{a}$, $\hat{a}$ be solutions.*
We call $\hat{a}$ maximal *or* (Pareto) optimal *iff no $\dot{a}$ exists such that $\dot{a}$ dominates $\hat{a}$. The set containing all optimal solutions is called the* Pareto front*.*

3.3 Metric points

Definition 3. *A* metric point *(or a* point *in the metric space) is a vector of metric values which derives from applying a solution to M.*

Note that different solutions s, $\hat{s}$ can result in the same metric point, $M(s) = M(\hat{s})$.

3.4 Solving a Multi-Objective Optimization Problem

The result of solving a multi-objective optimization problem is its Pareto Front, *i.e.* the set of all Pareto optimal soltuions. As an algorithm produces one solution at a time, however, we specifiy its output as a sequence rather than a set. An exact MOO solver has the three properties:

Definition 4. *Specification of a MOO solver*
Given decision variables $\boldsymbol{X}$, metric functions $\boldsymbol{M}$ and constraints $\boldsymbol{C}$; the set A of all solutions for $\langle \boldsymbol{X}, \boldsymbol{M}, \boldsymbol{C} \rangle$. A MOO solver produces a sequence of solutions $\boldsymbol{O}$ such that

i) Soundness*: Every generated solution satisfies the constraints:*
$\forall a \in O : \; C(a)$
ii) Optimality*: Every generated solution is optimal:*
$\forall a \in O : \; a \; is \; Pareto \; optimal$
iii) Completeness*: Every optimal solution is generated:*
$\forall a \in A : \; a \; is \; Pareto \; optimal \Rightarrow a \in O$

Specifying a solver in terms of a sequence of solutions that it produces has an additional advantage: the Pareto front for a MOOP can be so large that a user cannot wait for all optimal solutions to be generated; or the user may simply wish to start assessing and exploring solutions as they are generated. Thus, solutions should be yielded as early as possible instead of deriving the entire output set (which may never be obtained).

The algorithms presented in section 4 all yield solutions while they compute. They are so called *anytime* algorithms, *i.e.* if terminated prematurely, the algorithms outputs are approximations of the correct answer. Note, however, that there is a difference between the three algorithms: all solutions yielded by the guided improvement algorithm or the adaptive ϵ-constraint method are guaranteed to be *sound* and *optimal*. In comparison, the opportunistic improvement algorithm can only guarantee *sound* solutions. While its output sequence approximates the Pareto front, it is not possible to determine if a solution is *optimal* unless the algorithm runs to completion.

4 Algorithms

In the following we describe the three algorithms studied in this paper. The first two algorithms, the GIA and the OIA utilize the definition of *dominance* though they do it in different ways. The third algorithm, the *adaptive ϵ-constraint method* (AEM), allows us to compare the GIA and OIA, which both use a SAT solver, to a completely different technique which builds on the use of state-of-the-art single-objective solvers.

4.1 Guided Improvement Algorithm

The pseudo-code of our guided improvement algorithm is listed in Algorithm 1. The key idea is to use the constraint solver not only to find solutions, but also – by augmenting the constraint with appropriate formulas – to search for solutions that dominate ones found already, or that occupy the same Pareto point.

We assume a function $buildFormula$ that converts a boolean function into a formula; this allows us to reuse our earlier definitions (of Pareto domination and equality). Thus the expression $\mathit{buildFormula}(\lambda x.\mathit{dominates}(x, s, M))$, for example, returns the formula whose solutions are those that dominate s with respect to the metric function vector M. Likewise, $\mathit{buildFormula}(\lambda x.\mathit{dominates}(s, x, M))$ returns the formula whose solutions are those dominated by s w.r.t. M.

In the case of our implementation of our GIA and the OIA of Gavanelli [11] and Lukasiewycz et al. [18], *buildFormula* is provided by the Kodkod relational model-finder [23, 24]. The relational part of Kodkod is fully described in Torlak's dissertation [23]. The arithmetic circuit construction done by Kodkod is based on standard techniques [1, 20], and is implemented in the class kodkod.engine.bool.TwosComplementInt. Kodkod is widely used as a backend solver for software engineering and program analysis tools, and its source code has been online for a few years (currently at http://alloy.mit.edu/kodkod/).

The algorithm repeatedly generates Pareto-optimal solutions as follows. First, it solves the constraint (using the solver function $SolveOne$ that returns a solution or $\emptyset$ if none exists). It then attempts to improve the solution by repeatedly solving for a new solution that dominates it. When no further dominating solution is found, the last solution is known to be on the Pareto front. All solutions that are Pareto-equal to it are then yielded in turn (this functionality is optional in our actual implementation). The constraint is then augmented to ensure that subsequent solutions are not dominated by the solutions generated in this last phase. This process is repeated until no further solutions are found.

A sample execution of this algorithm is depicted in Figure 1, which we now explain in more detail. The example problem has two metrics m_1 and m_2 which both should be maximized. The space of possible solutions $p1, \ldots, p7$ is shown in Figure 1a. A visual inspection reveals that the Pareto-optimal solutions are p_7 and p_4. The algorithm might discover these optimal solutions as follows.

After an initialization (line 1 in the pseudo code) we solve the constraint F (line 2) and get a solution s as a result. In Figure 1, s relates to p_1. Based on the metric values obtained, we now construct a formula $betterMetric$ that will

Algorithm 1: Guided Improvement Algorithm

input : *Constraint* F, *Metric* M

1 $\mathit{Formula}\ \mathit{notDominated} \leftarrow \mathit{true}$
2 $\mathit{Solution}\ s \leftarrow \mathit{SolveOne}(F)$
3 **while** $s \neq \emptyset$ **do**
4   **while** $s \neq \emptyset$ **do**
5         $s' \leftarrow s$
6         $\mathit{Formula}\ \mathit{betterMetric} \leftarrow \mathit{buildFormula}(\lambda x.\mathit{dominates}(x, s, M))$
7         $s \leftarrow \mathit{SolveOne}(F \wedge \mathit{betterMetric})$
8     $\mathit{Formula}\ \mathit{sameMetric} \leftarrow \mathit{buildFormula}(\lambda x.\mathit{equals}(x, s', M))$
9   **for** a *in* $\mathit{SolveAll}(F \wedge \mathit{sameMetric})$ **do**
10   **yield** a
11     $\mathit{notDominated} \leftarrow \mathit{notDominated} \wedge \neg \mathit{buildFormula}(\lambda x.\mathit{dominates}(s', x, M))$
12     $s \leftarrow \mathit{SolveOne}(F \wedge \mathit{notDominated})$

force the solver to find only solutions which dominate s (line 6). The formula ensures that a future solution $\dot{a}$ satisfies:

$$\bigvee_{j=1}^{q} \left(\left(\bigwedge_{k=1, k \neq j}^{q} m_k(\dot{a}) \geq m_k(s) \right) \wedge m_j(\dot{a}) > m_j(s) \right)$$

For the small example in Figure 1b, the formula ensures that p_2 and p_5 are no longer in the solution space.

Solving the constraint F in conjunction with $betterMetric$ leads to a new, "better" solution, in the example to p_3. We repeat the process of augmenting the constraint F with a $betterMetric$ formula until, at some point (cp. Figure 1e), no further solutions can be found. By construction of the formula, this guarantees that the last solution s' (p_7 in the example) is Pareto-optimal.

Using the metric values of this Pareto-optimal solution, we now find all solutions at the same point (cp. Figure 1f). This is done by building a $sameMetric$ formula (line 8) that restricts the solution space to only contain solutions which are (Pareto) equal to the optimal solution. If s' is a Pareto-optimal solution then the formula ensures that every future solution $\dot{a}$ satisfies:

$$\bigwedge_{j=1}^{q} m_j(\dot{a}) = m_j(s')$$

All the equal solutions are yielded (line 9, 10).

Before we restart the procedure of "climbing up" to a Pareto-optimal solution, a new solution is needed from where the climbing can be started. In order to guarantee that this climbing will end at some, so far, undiscovered optimal solution, the *start solution* must not be dominated by any optimal solution already found. This is achieved with the help of a *notDominated* formula (line

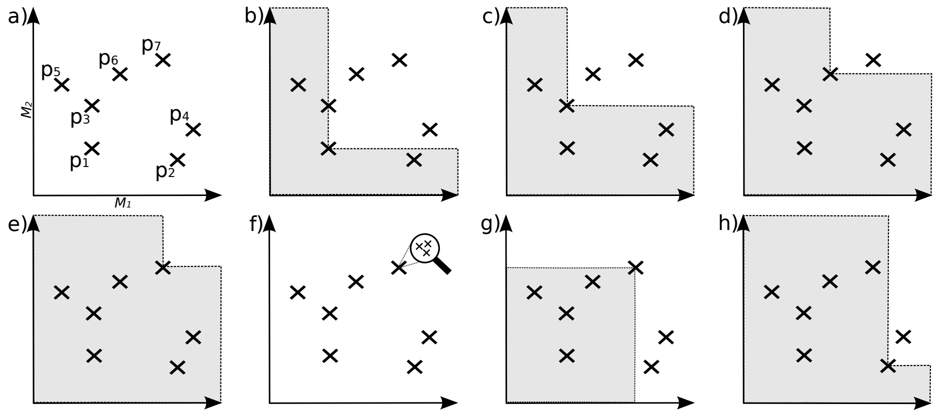


Fig. 1. GIA: Guided Improvement Algorithm; Gray areas are pruned from the solution space.

11) which ensures that a future solution $\dot{a}$ is better on at least one metric than the optimal solution s':

$$\bigvee_{j=1}^{q} m_j(\dot{a}) > m_j(s')$$

In Figure 1h we assume p_2 to be this new starting solution. The algorithm terminates when no new start solution can be found.

Assuming we only yield a single solution per Pareto point p (*i.e.* we ignore the loop in line 9), we can estimate the number of calls to the base-solver with $O(2p + 1 + y)$. For each Pareto point we have to invoke the solver twice; first to discover it, second to proof its Pareto-optimal property (an UNSAT call to the solver). An additional (UNSAT) call is needed to ensure that all points were found. y is a measure of how lucky the base-solver is when "climbing" towards Pareto points.

4.2 Opportunistic Improvement Algorithm

We implemented the *opportunistic improvement algorithm* that has been described in [11, 18] to work within our framework. Its pseudo code is shown in Algorithm 2. Again, it is easier to explain the code with the help of a small example which is shown in figure 2.

After some initialization and declaration (line 1 and line 2) we solve the constraint F (line 3) and get a solution, e.g. p_1 in figure 2b. Each solution s, that is found at some point, is added to the set S (line 5). Subsequently, the $filter()$ operation will remove all solutions in S which are now dominated by the newly added s. Thus, no solution in S is dominated by any other solution in S. Using the function $buildFormula$ we generate a formula to exclude solutions

Algorithm 2: Opportunistic Improvement Algorithm

input : *Constraint* F, *Metric* M
output: $Set\langle Solution\rangle$ S

1 $S \leftarrow \emptyset$
2 *Formula notDominated* $\leftarrow$ *true*
3 *Solution* $s \leftarrow SolveOne(F)$
4 **while** $s \neq \emptyset$ **do**
5 $\quad S \leftarrow filter(S.add(s))$
6 $\quad notDominated \leftarrow notDominated \wedge \neg buildFormula(\lambda x.dominates(s, x, M))$
7 $\quad s \leftarrow SolveOne(F \wedge notDominated)$
8 **return** S

dominated by s and build the conjunction with formula *notDominated* (line 6). We solve the constraint F in conjunction with *notDominated*. Figures 2b) to h) show these last steps over multiple iterations. In case no new solution can be found anymore, the OIA returns the set S and terminates.

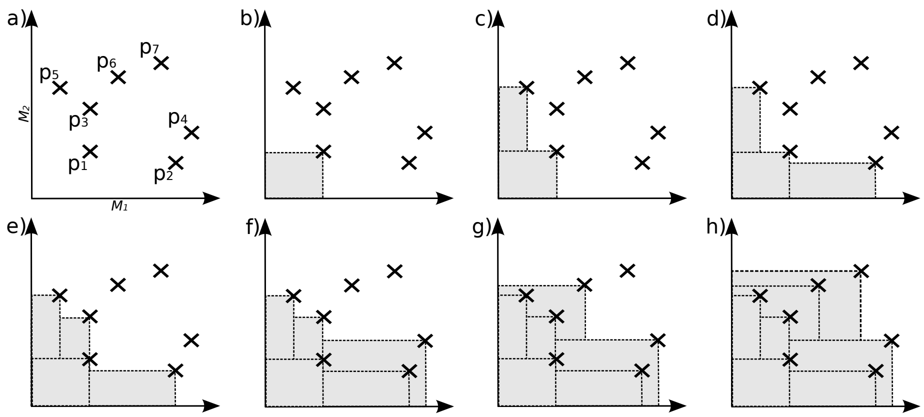


Fig. 2. OIA: Opportunistic Improvement Algorithm; Gray areas are pruned from the solution space.

The OIA as described in Algorithm 2 (and [18]) returns a set S of solutions where each Pareto-optimal point is represented by one optimal solution.We can estimate the number of calls to the base-solver with $O(p + 1 + x)$. The solver is invoked p times, once per Pareto point, plus one additional call to ensure all points were found. x represents the number of calls for non-optimal solutions.

The algorithm can be trivially modified to return all Pareto-optimal solutions, as is done in the description of the GIA above.

The implementation of the $filter()$ operation is not described in Lukasiewycz et al. [18]. In personal communication, Lukasiewycz has told us that it is the naïve approach. Our measurements of our filtering implementation (not reported in detail due to space limitations) show that it is never more than 5% of total runtime, and on larger problems is usually only 1% or 2%.

4.3 Adaptive epsilon-constraint method

Laumanns et al. [17] present an approach called *adaptive ϵ-constraint method* which uses a standard single-objective optimization solver like CPLEX [14] to iteratively discover Pareto-optimal solutions.

The idea of an *ϵ-constraint method* [13] is to only optimize a single metric while fixing all other metrics to some bound so they function as constraints. By systematically modifying the bounds on these additional constraints, it is possible to iteratively discover solutions on the Pareto front.

Laumanns et al. [17] point out that the original version of the *ϵ-constraint method* depends on a predefined constant δ which determines how the constraint bound for the fixed metric functions changes over iterations. Choosing this δ correctly is important; if it is to "too big", the method might not discover all Pareto-optimal solutions. If δ is chosen "too small", the method might become inefficient as too many runs of the single-objective solver are required.

The *adaptive ϵ-constraint methods* overcomes these problems of the original method and automatically adapts the bounds for the fixed objectives. We refer to the original paper for a detailed explanation of the method and the estimation of calls to the base-solver which grows exponentially with the number of objectives.

For our present purposes, it is only important that the *adaptive ϵ-constraint method* is an exact, general-purpose multi-objective solver. The code of Laumanns et al. [17], which we use in our comparison, was sent to us by Marco Laumanns.

5 Case Studies

Our interest in multi-objective optimization stems from a collaboration with a group of aerospace engineers who are interested in using it to analyze systems architecture decisions. From studying the kinds of models they are interested in writing (*e.g.*, [2, 12, 16, 21, 22]) we have discovered that they are essentially discrete multi-objective constraint satisfaction problems: each model has a finite set of decision variables, each ranging over a finite domain.

We have also identified two models that are characteristic of the kinds of models these aerospace engineers tend to write: (1) mission-mode decisions (*e.g.*, [12, 16, 22]), and (2) satellite launch scheduling (*e.g.*, [2, 21]). We now discuss these characteristic models.

NASA's Moon and Beyond Program. Since 2004 NASA has been planning to return to the moon, as a first step to future manned deep-space missions. In contrast to the original Apollo missions, these new plans involve astronauts staying on the moon for an extended period of time and exploring distant regions of the moon surface.

Our aerospace collaborators had previously modelled the mission-mode decisions of the Apollo flights (*e.g.*, [16, 22]), and are now working on a model of these future moon missions [12]. Examples of mission-mode decisions include whether to perform an earth-orbit rendezvous, what kind of fuel to use, etc. The metrics in this model are *weight* and *risk*.

NASA's Decadel Survey. NASA recently formulated a ten-year satellite launch plan (decadel survey) for the years 2010–2019 [19]. Our aerospace collaborators have made a post-hoc multi-objective model of these decisions [2, 21]. The challenge of this model is to find a launch schedule that maximizes the scientific value of the satellites for six different scientific communities (*i.e.*, there are six objectives), while respecting various resource limitations and launch-ordering constraints. Each satellite provides different levels of value to each scientific community, and of course each community would prefer to have their data sooner rather than later.

6 Algorithm Comparison

This section compares the performance of the three algorithms discussed above (guided improvement algorithm, opportunistic improvement algorithm, and adaptive ϵ-constraint method). All three algorithms are compared on the multi-objective knapsack problem, which is also used by Laumanns et al. [17]. The guided improvement algorithm and opportunistic improvement algorithm are also compared using n-Queens and n-Rooks (explained below), as well as the two aerospace case studies discussed above. All alogrithms are restricted to only find one solution per Pareto point.

We have not yet evaluated the adaptive ϵ-constraint method on these other problems because it is implemented in a different framework using a different formalism, and reads a different input file format. There is a need for the community to design standard formalisms, frameworks, and input file formats to make it easier to compare algorithms.

To the best of our knowledge, this is the first paper to compare combinatorial multi-objective algorithms invented by different authors. We also use more challenge problems than previous related papers: for example, Junker [15] and Ehrgott and Ruzika [7] do not have any empirical evaluation; Gavanelli [11] considers only the multi-objective knapsack problem; Laumanns et al. [17] considers both the multi-objective knapsack problem and the Bi-Binary-Value problem; Lukasiewycz et al. [18] considers the n-Queens problem and a problem from the area of system level synthesis.

6.1 n-Queens

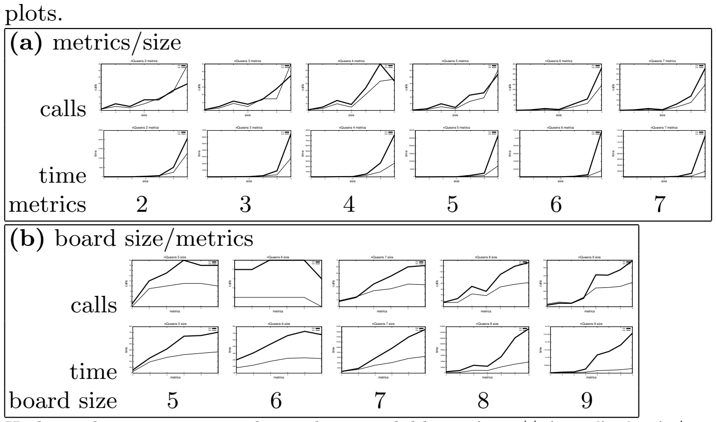
n-Queens is a classic NP-hard problem. The constraint is to place n Queens on to an $n \times n$ chess board so that none of them can strike each other. Following

Lukasiewycz et al. [18], we generate objective functions for n-Queens as follows: assign a random integer between 0 and n to each square on the board; sum the scores of each square with a Queen on it. Multiple such objective functions can be generated by generating a new set of random square-scores for each new objective function.

Figure 3 shows that the opportunistic improvement algorithm has better performance, both in terms of calls to the base-solver and in terms of time, than the guided improvement algorithm on the n-Queens problem for all combinations of board size and number of metrics that we examined.

Figure 3 also shows, interestingly, that for larger boards and more metrics that the guided improvement algorithm does not make many more calls to the base-solver, but takes much longer to solve: in other words, not all calls to the base-solver are equal.

Fig. 3. n-Queens comparison. guided improvement algorithm performance is represented by the bold line; opportunistic improvement algorithm performance is represented by the faint line. Lower values are better. y-axes are not comparable between plots.



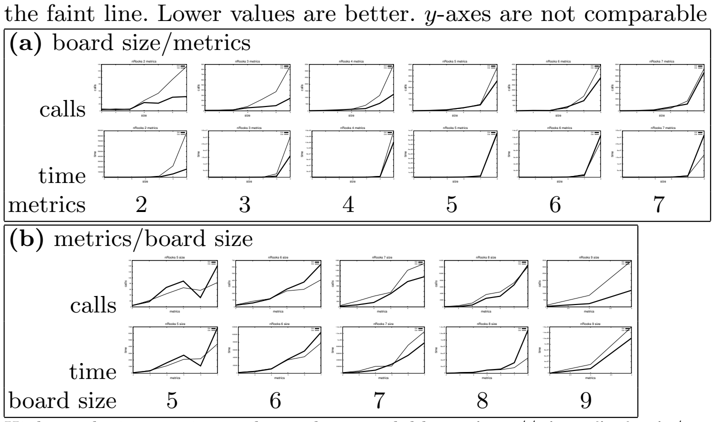
High resolution version and raw data available at: http://sdg.csail.mit.edu/moolloy/

6.2 n-Rooks

The n-Rooks problem is a relaxation of the n-Queens problem where the pieces are Rooks (castles) instead of Queens. We generate objective functions for n-Rooks in the same way as for n-Queens.

Figure 4 shows that the guided improvement algorithm almost always makes fewer calls to the base-solver than the opportunistic improvement algorithm does, and that the guided improvement algorithm usually – but not always – takes less time. Again, some calls to the base-solver are more expensive than others. Figure 4 also shows that the guided improvement algorithm takes significantly less time for n-Rooks instances with larger boards and two or three objectives.

Fig. 4. n-Rooks comparison. guided improvement algorithm performance is represented by the bold line; opportunistic improvement algorithm performance is represented by the faint line. Lower values are better. y-axes are not comparable between plots.



High resolution version and raw data available at: http://sdg.csail.mit.edu/moolloy/

6.3 Multi-objective KnapSack

The multi-objective 0/1 KnapSack problem is a standard benchmark problem to evaluate the performance of multi-objective solvers. We use this problem to compare the performance of the opportunistic improvement algorithm and the guided improvement algorithm to the adaptive ϵ-constraint method. This problem has also been used by Laumann et al. in their original paper [17].

In the single-objective 0/1 Knapsack problem, a set of n items is given. Each item is associated with a *profit* value and a *weight* value. The goal is to select those items which maximize the sum of profits while the sum of their weight values is less than some constant. Extending this problem to the multi-objective case is simply done by defining multiple profit values and weight values for each item, respectively.

We encode the problem using boolean variables $x_1, \ldots, x_n$, where each variable denotes whether an item is selected or not. We define m profit functions $P_j(x_1, \ldots, x_n)$, $1 \leq j \leq m$, such that each x_i is associated with m profit values. Furthermore, we define m weight constraints $W_j(x_1, \ldots, x_n)$, $1 \leq j \leq m$, sucht that each x_i is associated with m weight values w_{ij}. For each W_j, the sum $\sum_{i=1}^{n} w_{ji} * x_i$ has to less or equal to $\lfloor \frac{\sum_{i=1}^{n} w_{ij}}{2} \rfloor$. All profit and weight values are randomly chosen integers between 10 and 100.

Table 1 shows the results for different numbers of variables and objectives. We find that the guided improvement algorithm outperforms the opportunistic improvement algorithm on all relevant problem sizes. We observe a significant difference in the runtime especially on those problems were the OIA invokes the solver much more often than the GIA. The data indicates that a higher number of SAT calls of the OIA outweighs the costs of the GIA's UNSAT calls (usually SAT calls are assumed to be less expensive than UNSAT calls).

Table 1. Results for the KnapSack problem. n is the number of variabes, m the number of metrics. *Solns* is the number of Pareto points. t shows the solving time for the three different algorithms. c show how often the base-solver was called. For the *OIA* and the *GIA* the number of (SAT, UNSAT) calls is given in parenthesis. Symbols: † means the algorithm stopped prematurely with an error-code; ⋆ means that the algorithm did not terminate within 48 hours; Machine used: Pentium 4 dual core, 3GHz, 2GB Ram; ILOG CPLEX version 11.0. Time measured as clock time.

$(n \times m)$	*Solns*	t_{OIA}	t_{GIA}	t_{AEM}	c_{OIA}		c_{GIA}		c_{AEM}
(10×2)	4	<1 s	<1 s	<1 s	17	(16, 1)	15	(10, 5)	9
(15×2)	3	6 s	**1 s**	2 s	64	(63, 1)	17	(13, 4)	9
(20×2)	4	31 s	9 s	**2 s**	103	(102, 1)	30	(25, 5)	9
(25×2)	16	22.8 m	6.8 m	**6 s**	279	(278, 1)	73	(56, 17)	35
(30×2)	33	3.1 h	30.8 m	**9 s**	516	(515, 1)	131	(97, 34)	73
(10×3)	6	2 s	**1 s**	3 s	65	(64, 1)	24	(17, 7)	43
(15×3)	28	24 s	**17 s**	27 s	142	(141, 1)	78	(49, 29)	317
(20×3)	71	15.6 m	7.6 m	**2.8 m**	496	(495, 1)	195	(123, 72)	1831
(25×3)	207	29.6 h	11.5 h	**23 m**	1811	(1810, 1)	549	(341, 208)	9571
(10×4)	37	**3 s**	4 s	23.9 m	69	(68, 1)	93	(55, 38)	10545
(15×4)	52	1 m	**42 s**	42.8 m	307	(306, 1)	129	(76, 53)	19507
(20×4)	156	2.5 h	**58.2 m**	†	1273	(1272, 1)	387	(230, 157)	†
(25×4)	813	⋆	**45 h**	†	⋆		1938	(1124, 814)	†

The adaptive ϵ-constraint method is the performance leader on all problem sizes with 2 or 3 metrics, even if the number of solver calls is comparably high. However, for problems with 4 metrics the guided improvement algorithm appears to be best: it was the only algorithm to terminate normally on the largest problem (25×4), and was also noticably faster on smaller problems with four metrics.

6.4 Lunar Lander and Decadel Survey Case Studies

Table 2 characterizes the Lunar Lander [12] and Decadel Survey [2] models, and the performance of the guided and opportunistic improvement algorithms on these models. The guided improvement algorithm performs better in both cases: about twice as fast for the Lunar Lander model, and about four times faster for the Decadel Survey model. Interestingly, both algorithms make almost the same number of base-solver calls for the Lunar Lander model: in this case it seems that the opportunistic improvement algorithm calls are harder than the guided improvement algorithm calls, which is the opposite of what we observed on the n-Rooks micro-benchmark.

	Decadel [2]		Lunar [12]	
Metrics	6		2	
Decisions	17		20	
Size of state space	5.05 $\times 10^{17}$		2.96 $\times 10^{10}$	
Pareto points	67		36	
Pareto solutions	67		1683	
Calls (for Pareto points)	705	396	158	151
Time (for Pareto solutions)	22.8m	**5.9m**	62s	**29s**
	OIA	GIA	OIA	GIA

Table 2. Characterization of and performance results for Lunar Lander and Decadel Survey case studies.

6.5 Summary of Findings

The findings of the experiments described above are summarized in Table 3. No single algorithm is best for all kinds of problems. However, our guided improvement algorithm appears to be the best for the kinds of many-objective problems that our aerospace collaborators are interested in solving. This conclusion is supported by three findings:

1. Our guided improvement algorithm performed better than the opportunistic improvement algorithm on the two aerospace case study problems.
2. Our guided improvement algorithm performed better than the opportunistic improvement algorithm on the n-Rooks problem, and worse on the n-Queens problem. The n-Rooks problem is more similar to the kinds of problems our aerospace collaborators are interested in because its constraints are not NP-hard. Both the Lunar Lander and the Decadel Survey case studies have relatively easy constraints: more like n-Rooks than n-Queens.
3. Our guided improvement algorithm performed better than the adaptive ϵ-constraint method on the knapsack problem with a higher number of metrics. The Decadel Survey case study has six metrics, and is the harder of the two case studies.

Problem	**Best Algorithm**
n-Queens	OIA
n-Rooks	GIA
Lunar Lander	GIA
Decadel Survey	GIA
Knapsack, 4 metrics	GIA
Knapsack, 2 or 3 metrics	AϵCM

Table 3. Summary of Findings

6.6 Threats to Validity

Internal validity refers to whether the experiments support the findings. External validity refers to whether the findings can be generalized.

Internal Validity. Threats to the internal validity of these experiments include: correctness of the algorithm implementations; fairness of the opportunistic improvement algorithm implementation; Java virtual machine startup time; flaws in the adaptive ϵ-constraint method code sent to us by Laumanns; and veracity of timings.

GIA versus OIA Comparison. We mitigated the correctness and fairness threats by implementing both the guided improvement algorithm and opportunistic improvement algorithm with the same library of subroutines, and ran them with the same SAT solver (MiniSAT [3]) on the same hardware (3GHz quad-core machine with 4GB RAM).

The opportunistic improvement algorithm requires an extra filtering subroutine that is not used by the guided improvement algorithm. In other experiments, not detailed in this paper, we have never observed this filtering overhead to take more than 5% of the execution time. For larger problems it is typically 1% or less of total runtime.

To assess correctness of our guided improvement algorithm and opportunistic improvement algorithm implementations we performed manual code reviews, wrote test cases, manually inspected the output of some small problems, and mechanically compared the output of the OIA and GIA after each run.

Most Java virtual machines have some startup costs associated with class loading and compilation. To mitigate these factors we always used a warm-up run and ran with a heap size larger than the programs need but smaller than available on the machine (a 1GB heap on a 4GB machine).

We measured clock time rather than CPU time because it is more convenient to measure the former in Java programs. Clock time and CPU time can diverge significantly for multi-threaded programs on multi-core hardware, or if the hardware is heavily taxed with other programs. Both of the algorithms we evaluated are single-threaded and we ran them on a lightly loaded quad-core machine. For these reasons, we believe that clock time is a reasonable approximation of CPU time in these experiments.

GIA verses adaptive ϵ-constraint method Comparison. We discovered a correctness problem in the code originally sent to us by Laumanns: it sometimes reports non-optimal solutions. It appears that its results are a super-set of the Pareto front. We reported this problem to Laumanns, who has subsequently corrected it (and may have been aware of it independently of our discovery). This problem does not have a significant performance impact though. It can be easily worked around by adding a post-processing filter, as is required by the opportunistic improvement algorithm. As we saw for the OIA, such a filter adds a negligible cost to the overall algorithm runtime, and so would not have changed the results significantly.

In order to work across a number of versions of CPLEX the code that Laumanns sent us communicates with CPLEX using temporary files. The code can be modified to communicate with a particular version of CPLEX directly. We chose to not modify the code that Laumanns sent us, in order to avoid introducing any new faults to it.

Writing these temporary files only affect a constant factor in the algorithm runtime: they do not change the complexity of the algorithm. The adaptive ϵ-constraint method is exponential in the number of metrics the problem has [17]. Our results show that the guided improvement algorithm outperforms the adaptive ϵ-constraint method on problems with a higher number of metrics. Due to the algorithmic complexity, this conclusion is likely to remain true independently of the constant factors involved in the implementation of the adaptive ϵ-constraint method. The constant factors will likely only change the value of 'higher' at which the GIA overtakes the adaptive ϵ-constraint method— if changing the constant factors has any effect at all for larger problems.

We performed the GIA versus adaptive ϵ-constraint method comparison on a different machine than the GIA versus OIA comparison due to CPLEX licensing restrictions. Our findings do not depend on comparing results across machines. The measurements reported in Table 1 were all taken on the CPLEX-licensed machine. The CPLEX-licensed machine was a 3GHz dual-core Pentium 4 with 2GB of RAM.

External Validity. External validity is concerned with whether the findings can be generalized beyond the specific experiments conducted. Our main finding is that our guided improvement algorithm is better than the opportunistic improvement algorithm and adaptive ϵ-constraint method for the kinds of real-world problems that our aerospace collaborators are interested in solving.

It is possible that the two aerospace case studies we used in this paper are not representative of the kinds of problems our collaborators wish to solve. We mitigated this concern by studying many of the problems our collaborators are working on, over a period of three years (from May 2006 to present), and choosing the most challenging ones for this paper. Our collaboration has been acknowledged in two PhD dissertations [16, 22] from their group, and we have interacted with over half a dozen other students in the group.

It is possible that the real-world problems that our collaborators want to solve are somehow different from real-world problems that other groups want to solve. We have not taken any substantial steps to mitigate this concern. We are not aware of any central repository of MOCO problems that could be studied for this purpose. (We are aware of such respositories for first-order theorem proving, SAT, SMT, and CSP problems.)

7 Conclusion

This paper introduced the *guided improvement algorithm* for exact solving of combinatorial multi-objective optimization problems and showed that it is both novel and useful. Novelty was demonstrated by an overview of the literature. Utility was demonstrated primarily on two real-world aerospace case studies, with supporting experiments on three micro-benchmarks (n-Queens, n-Rooks, and the multi-objective knapsack problem). Our guided improvement algorithm appears to be better than the opportunistic improvement algorithm [11, 18] and the adaptive ϵ-constraint method for the kinds of real-world problems that our aerospace collaborators are interested in solving.

To the best of our knowledge this is the first paper to compare general-purpose MOCO algorithms from different authors.

We found that each algorithm had at least one context in which it performed the fastest. The opportunistic improvement algorithm performed the best for the n-Queens problem (with the caveat that we did not run the adaptive ϵ-constraint method on n-Queens). The adaptive ϵ-constraint method performed the best on knapsack problems with two or three objectives. Our guided improvement algorithm out-performed the opportunistic improvement algorithm on all problems except n-Queens, and it outperformed the adaptive ϵ-constraint method on knapsack problems with four objectives.

The results of the knapsack evaluation suggest that our guided improvement algorithm might outperform the adaptive ϵ-constraint method for the Decadel Survey case study, which has six objective functions. Similarly, the knapsack results suggest that the adaptive ϵ-constraint method might be the best for the Lunar Lander case study, which has only two objectives.

Our empirical results confirm that considering the complexity of these algorithms in terms of the number of base-solver calls they make is useful – although not always definitive. We observed cases where the cost of a base-solver call varied significantly. We also observed cases where the adaptive ϵ-constraint method made an order of magnitude more base-solver calls than the other algorithms but still returned in the least amount of time.

There is a need for the MOCO community to develop standard formalisms, file formats, and challenge problems — as is already done in the CSP, SAT, SMT, theorem-proving, and other communities.

Acknowledgements

Many people have participated in helpful discussions of this work, and we are grateful for their time and insights; in alphabetical order: Felix Chang, Justin Colson, Ed Crawley, Greg Dennis, Arthur Guest, Wilfried Hofstetter, Andrew Yi Huang, Eunsuk Kang, Ben Koo, Ben Kuhn, Maokai Lin, Gustavo Pinheiro, Rob Seater, Theo Seher, Bill Simmons, Dan Sturtevant, Tim Sutherland, Emina Torlak, Olivier de Weck, and Brian Williams. We thank Zhaohui Fu, Marco Laumanns, Martin Lukasiewycz and Sharad Malik for providing us their code and examples, and for helpful discussions of their work.

This research was funded in part by the National Science Foundation under grant 0438897 (SoD Collaborative Research: Constraint-based Architecture Evaluation), and by the Air Force Research Laboratory (AFRL)/IF and Disruptive Technology Office (DTO) in the National Intelligence Community Information Assurance Research (NICIAR) Programme (ConfigAssure: Dynamic System Configuration Assurance for National Intelligence Community Cyber Infrastructure). A research scholarship was provided by the University of Paderborn.

Bibliography

[1] Multiplication ALU. URL http://en.wikipedia.org/wiki/Multiplication_ALU.

[2] Justin M. Colson. System architecting of a campaign of earth observing satellites. Master's thesis, MIT, 2008. Advised by Ed Crawley.

[3] Niklas Een and Niklas Sörensson. An Extensible SAT-solver. In *Proc. SAT*, 2003.

[4] Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4): 425–460, 2000.

[5] Matthias Ehrgott and Xavier Gandibleux. Multiobjective combinatorial optimization: theory, methodology, and applications. In Matthias Ehrgott and Xavier Gandibleux, editors, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Survey*, volume 52 of *International Series in Operations Research and Management Science*, pages 369–444. Kluwer Academic Publishers, Boston, MA, 2002. ISBN 1-4020-7128-0.

[6] Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. In Christian Blum, Maria José Blesa Aguilera, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*. Springer-Verlag, 2008. ISBN 978-3540782940.

[7] Matthias Ehrgott and Stefan Ruzika. Improved ϵ-constraint method for multiobjective programming. *Journal of Optimization Theory and Applications*, 138(3):375–396, 2008. doi: 10.1007/s10957-008-9394-2.

[8] Matthias Ehrgott, José Figueira, and Xavier Gandibleux. Special issue on multiple objective discrete and combinatorial optimization. *Annals of Operations Research*, 147(1), October 2006.

[9] José Figueira, Salvatore Greco, and Matthias Ehrgott, editors. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer-Verlag, 2005.

[10] Xavier Gandibleux and Matthias Ehrgott. 1984–2004 — 20 years of multiobjective metaheuristics. but what about the solution of combinatorial problems with multiple objectives? In Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors, *Proc. 3rdEvolutionary Multi-Criterion Optimization*, volume 3410 of *LNCS*, pages 33–46, Guanajuato, Mexico, March 2005. ISBN ISBN 3-540-24983-4.

[11] Marco Gavanelli. An algorithm for Multi-Criteria Optimization in CSPs. In Frank van Harmelen, editor, *Proc. 15thEuropean Conference on Artificial Intelligence*, Lyon, France, July 2002. IOS Press.

[12] Arthur Guest. Lunar lander revisited. Master's thesis, MIT, 2009. In preparation. Advised by Ed Crawley.

[13] Y. Haimes, L. Lasdon, and D. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:296–297, 1971.

[14] ILOG. CPLEX v11.0. URL http://www.ilog.com/products/cplex/.

[15] Ulrich Junker. Outer branching: How to optimize under partial orders?, Undated. URL http://wikix.ilog.fr/wiki/pub/Main/UlrichJunker/opo.pdf. Preliminary versions published at MOPGP'06 and M-PREF'06.

[16] H.-Y. Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, MIT, 2005. Advised by Edward Crawley.

[17] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006.

[18] Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Solving multiobjective pseudo-boolean problems. In *Proc. SAT*, pages 56–69, Lisbon, Portugal, May 2007.

[19] National Research Council Space Studies Board. *Earth Science and Applications from Space: National Imperatives for the Next Decade and Beyond*. National Academies Press, 2007.

[20] Behrooz Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[21] Theodore K. Seher. Decadel survey revisited. Master's thesis, MIT, 2009. In preparation. Advised by Ed Crawley.

[22] Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, MIT, 2008. Advised by Edward Crawley.

[23] Emina Torlak. *A Constraint Solver for Software Engineering: Finding Models and Cores of Large Relational Specifications*. PhD thesis, MIT, 2008. Advised by Daniel Jackson.

[24] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In Orna Grumberg and Michael Huth, editors, *Proc. 13^{th} TACAS*, volume 4424 of *LNCS*, pages 632–647, Braga, Portugal, March 2007. Springer-Verlag.

[25] E. L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1993.

[26] Jyrki Wallenius, James S. Dyer, Peter C. Fishburn, Ralph E. Steuer, Stanley Zionts, and Kalyanmoy Deb. Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Management Science*, 54(7):1336–1349, July 2008.