

Toward Adaptive Cooperation between Global and Local Solvers for Continuous Constraint Problems *

Yi Shang, Yungchun Wan, Markus P.J. Fromherz, and Lara S. Crawford

Xerox Palo Alto Research Center

3333 Coyote Hill Road

Palo Alto, CA 94304

{yshang,jwan,fromherz,lcrawford}@parc.xerox.com

Abstract

Search methods for solving continuous constraint problems can be broadly divided into two categories: global search and local search methods. Global search methods spend a great amount of effort exploring the global search space, whereas local search methods focus on converging to local optimal solutions. Although these methods alone work well on many problems, there are many others that will benefit from using cooperative local and global search strategies in a problem solving process. In this paper, we present some alternatives of combining existing local and global methods as cooperative solvers for constraint problems. To demonstrate the benefits of the combined solvers, we compare their performance with individual methods on a series of continuous constraint problems with varying ratio of constraints to variables. The test problems are modeled after control problems in robotics and are very flexible, capable of generating a variety of problems with different characteristics. Our experimental results show that local search methods are best for weakly constrained problems, whereas combining global and local methods is better for highly constrained, harder problems. As a consequence, the balance between global and local search in a combined solver should be adaptive to problem characteristics such as the ratio of constraints to variables.

1 Introduction

Many important real-world software problems in domains such as planning, control, reconfiguration, and fault diagnosis can be regarded as constrained optimization problems [Bondarenko *et al.*, 1998]. These optimization problems are often best solved in the continuous domain, either because the underlying physical system is continuous or because discrete implementations exhibit abrupt changes in outputs for small changes in inputs. However, effective constraint-based techniques must handle the complexity of real-world continuous

constraint problems by dynamically adapting solvers to the structure of the problem. Toward this end, this paper extends work on complexity analysis for discrete constrained optimization/satisfaction problems to continuous problems.

In the past decade, significant progress has been made in understanding problem complexity of discrete constraint satisfaction and combinatorial optimization problems, such as satisfiability (SAT), planning, graph coloring, and the traveling salesman problem (TSP) [Hogg *et al.*, 1996a]. One particularly well-studied phenomenon is the *complexity phase transition* [Hogg *et al.*, 1996b]. For example, in solving randomly generated SAT problems, the average time to find a solution or determine that none exists is short when the ratio of clauses to variables is small (i.e., the problem is weakly constrained) as well as when this ratio is large (where the problem is highly or overconstrained), while solving time is largest in the intermediate case. This phenomenon has been observed for a variety of complete and incomplete search algorithms, including the Davis-Putnam method [Selman *et al.*, 1996] and GSAT [Selman *et al.*, 1992; Yokoo, 1997].

Similar complexity studies are useful in the continuous domain for better understanding the difficulty of continuous constraint problems as well as the computational complexity of various solving techniques. Regarding continuous constraint satisfaction problems, many generic and problem-specific methods have been developed in the past. They include various mathematical programming methods, direct search methods, evolutionary methods, and simulated annealing. These search methods can be broadly divided into two categories: global and local search methods. Global search methods expend a great amount of effort exploring the global space, whereas local search methods focus on converging to local optimal solutions. Although these methods alone work well on many problems, there are many other problems that will benefit from using cooperative local and global search strategies in a problem solving process.

In this paper, we present a general framework for combining local and global methods as cooperative solvers and explore alternative implementations. To demonstrate the benefits of the combined solvers, we compare their performance with individual methods on a series of continuous constraint problems. Since the ratio of constraints to variables (constraint ratio) has been identified as a key parameter of complexity phase transition [Shang *et al.*, 2001], in this paper we

*Published in CP'01 Workshop on Cooperative Solvers in Constraint Programming, Dec. 2001. This work has been supported in part by DARPA under contract F33615-01-C-1904.

Table 1: Existing continuous optimization methods and examples of their implementations.

Methods	Examples	Category
SQP	fmincon in Matlab	Local search
Quasi-Newton	fminunc in Matlab	Local search
Interior Point	LOQO	Local search
Nelder-Mead	fminsearch in Matlab	Local & global search
SA	ASA	global search

compare the performance of various solvers based on continuous constraint problems with varying constraint ratio.

The rest of this paper is structured as follows. In Section 2, we briefly discuss representative search algorithms on continuous constraint problems and their implementations used in our experiments. In Section 3, we review previous work on combining global and local search methods and present general frameworks for cooperative solvers. In Section 4, we present the test problems for comparing the performance of different methods. In Section 5, we show experimental results of various individual and cooperative solvers on problems with different constraint ratios. Finally, in Section 6, we conclude the paper and discuss future work.

2 Search Algorithms

Different search algorithms have different complexity behaviors in solving continuous constraint problems. In this section, we review some popular methods, including sequential quadratic programming methods, interior point methods, direct search methods, and simulated annealing. Table 1 summarizes the methods and examples of their implementations.

2.1 Sequential Quadratic Programming

Sequential quadratic programming (SQP) methods represent the state-of-the-art in nonlinear programming methods. Based on the work of Biggs, Han, and Powell, an SQP method mimics Newton’s method for constrained optimization [Powell, 1983]. It is an iterative method starting from some initial point and converging to a constrained local minimum. At each iteration, one solves a quadratic program (QP) that models the original nonlinear constrained problem at the current point. The solution to the QP is used as a search direction to find an improving point, which is used in the next iteration. This iteration is repeated until an optimal or feasible solution is found (for optimization or satisfaction problems, respectively). The SQP solver used in our experiments is fmincon from the Matlab Optimization Toolbox.

2.2 Interior-Point Method

Interior-point methods are also among the most successful linear and quadratic programming methods. These methods search along an arc in the interior of the feasible set (as contrasted with simplex methods). The technique for choosing the search direction at each step is a modification of Newton’s method [Nocedal and Wright, 1999]. LOQO is a very efficient interior-point method that extends its success to non-convex nonlinear programming [Vanderbei and Shanno, 1999]. The

major modifications to its quadratic programming version include a new merit function and an altered search direction to ensure that a descent direction for the merit function is obtained. For convex problems a globally optimal solution is obtained. For non-convex ones a locally optimal solution near a given initial solution is found. Numerical comparisons with other implementations show that the method is efficient and has the promise of greatly reducing solution times on some classes of problems [Dolan and Moré, 2001]. The LOQO solver used in our experiments is the trial program of version 6.00. It can work only with problems of limited sizes and is called within the AMPL environment [Fourer *et al.*, 1993].

2.3 Nelder-Mead Method

The Nelder-Mead (NM) algorithm, first published in 1965 [Nelder and Mead, 1965], falls in the class of *direct search methods*, which minimize a scalar-valued nonlinear function of real variables using only function values, without any derivative information (explicit or implicit). The NM algorithm is the most popular of the direct search methods. It is particularly effective for multi-dimensional unconstrained optimization [McKinnon, 1998; Lagarias *et al.*, 1998].

The NM algorithm is an iterative method. To minimize an n -dimensional objective function, it starts with a simplex in n -D space, a polyhedron with $n + 1$ vertices. At each iteration, the method evaluates the objective function at one or more trial points whose location depends on the relative values of the function at the vertices and at earlier trial points. The simplex is altered by reflection, expansion, or contraction in order to find a new point improving the worst vertex, i.e., the vertex with the largest objective value. At the end of each iteration, a new simplex is created in which the worst vertex has been replaced.

The NM algorithm can perform either global or local search, depending on the size of the initial simplex. When doing local search, previous work shows that the NM method usually has slow convergence compared to derivative-based local search methods, such as SQP, in solving smooth functions. The NM solver used in our experiments is fminsearch from the Matlab Optimization Toolbox.

2.4 Simulated Annealing

Simulated annealing (SA) is among the most powerful algorithms for nonlinear and stochastic systems. It is inspired by the physical process of annealing (e.g., of steel). At each step in the algorithm, a random displacement of the current point is proposed. If the “energy” (cost) of this new point is lower than that of the old point, the new point is accepted. If the new energy is higher, the point is accepted probabilistically, with probability dependent on a “temperature” parameter. Adaptive Simulated Annealing (ASA) [Ingber, 2001] is an improved implementation of SA with a sophisticated annealing schedule and the possibility of re-annealing. ASA’s annealing schedule is faster than other implementations, and the algorithm has been applied successfully to many applications.

The ASA solver we used in our experiments is version 22.11. ASA has over 100 options and we simply use its default

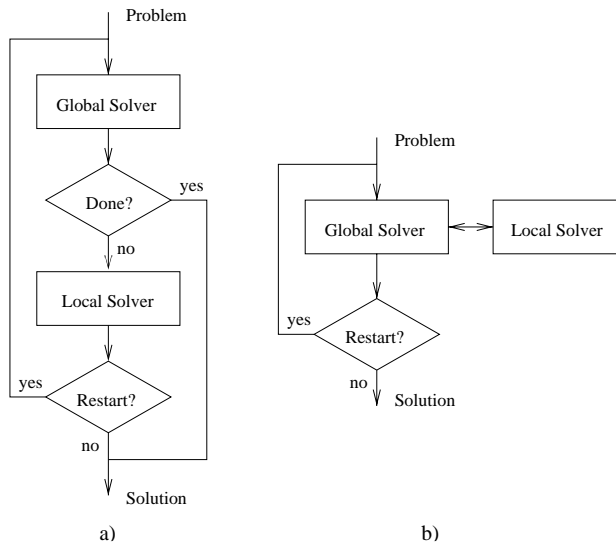


Figure 1: Two approaches for combining global and local solving: a) in series, b) as refinement.

settings with a few exceptions: the option `Limit.Generated`, which controls the maximum number of states generated before quitting, is set to a constant as described in the next section; the option `Curvature_0` is set to `TRUE` to reduce the number of function evaluations, as we do not use curvature information. Whenever all constraints are satisfied, the objective function evaluation routine terminates ASA by setting an immediate exit flag. Otherwise, ASA keeps on improving the penalty function. Since ASA is written in C, we ran the experiments in Matlab via a Matlab-to-C interface called ASAMIN [Sakata, 2001] (version 1.15).

3 Cooperative Solvers

Various forms of meta-heuristics have been proposed to combine global and local search heuristics, for both discrete and continuous optimization problems. Generally speaking, there are two approaches to combining global and local search: global and local search in series (Figure 1a) and local search as refinement for global search (Figure 1b). The goal of the former approach is to build on the individual strengths of global and local solvers: the global solver moves towards feasible regions or global optimal solutions, from which the local solver converges quickly to local minima. In other words, the two solvers cooperate by one contributing information about the global structure and the other providing fast local convergence. The idea of the second approach is for the global solver to use a local solver to improve its intermediate points to true local optima. In other words, the global solver is the primary algorithm; however, any point generated during search is first moved to a local optimum with help of the local solver. In this paper we focus on the first approach.

Researchers have experimented with many combinations of global and local search methods. For example, in [Hart, 1994], Hart uses genetic algorithms for global search and his choices for local search are random local search, a conjugate gradient method, and stochastic approximation. The GLO

(Global Local Optimizer) software uses a genetic method as the global technique and variable metric nonlinear optimization as the local technique [Murphy and Baker, 1995]. LGO (Lipschitz-Continuous Global Optimizer) integrates several global (adaptive partition and random search based) and local (conjugate directions type) strategies [Pinter, 1996]. In [Kitts, 1997], an enumerative strategy (uniform random sampling) is combined with greedy strategy (hillclimbing). Most of these approaches have been applied to unconstrained problems. Much similar work exists for combinatorial problems as well. For example, in [Martin, 1996], a meta-heuristic called “Chained Local Optimization” embeds deterministic local search techniques into simulated annealing for traveling salesman and graph partitioning problems. Genetic algorithms have also been combined with local searches for combinatorial problems [Muhlenhein *et al.*, 1988].

There is less work on cooperative solving of continuous constraint satisfaction problems (CSPs). In our previous work on continuous CSPs, we have observed that local search methods such as SQP combined with random restarts work best on weakly constrained problems where feasible regions are large or plentiful [Fromherz *et al.*, 2001; Shang *et al.*, 2001]. However, in highly constrained problems where feasible solutions are rare and the search space is rugged, local search methods need to have a very good starting point, not far away from a feasible solution, to be successful. In this circumstance, random restart usually fails to provide good enough starting points and global search methods provide the most benefit. Conversely, global search methods, such as simulated annealing and to some extent the Nelder-Mead method, employ systematic approaches to explore the global search space and are more effective in discovering and utilizing global features to find solutions. They are generally slow in converging, however, and later phases of search should be replaced by local solving using information such as gradient and curvature.

From these observations, it appears that the cooperation between global and local solvers should adapt to problem characteristics such as the constraint ratio. In particular, when combining global and local solving, a primary consideration is how much work to assign to each part: running the global solver longer may provide a better starting point for the local solver, but this effort may be wasted if the problem is weakly constrained, when the local solver is much better at converging. Conversely, increasing the global solver’s iteration limit for larger or harder problems should generate better starting points for local solving, reducing the number of necessary restarts in the process. With a view towards making this choice adaptively, we have conducted a number of experiments with different termination criteria for the global solver. For some algorithms and known problems, running the global solver for a *fixed maximum number of iterations* may be a sufficient solution. If the solver fails to find a solution within that time, its best point is used as the starting point for the local solver. This entire process is restarted using random restart until a solution is found. As a second tradeoff technique, we increase the iteration limit *proportionally* to the problem’s constraint ratio. Finding the best relation between problem parameters and iteration limit is non-trivial; for this paper, we just use a simple linear relation between constraint

ratio and iteration limit to illustrate the potential benefits.

A class of alternative techniques stop the global solver not after a certain number of iterations, but based on feedback from the search process. For example, one may stop the search when the solver’s *rate of improvement* (as measured by improvements in the objective function with each iteration) is below a given threshold (e.g., a certain percentage of the initial function value). Another technique is to stop when the solver actually is *close to a feasible region*. The distance may be represented by some merit function, and the algorithm is stopped when the value of this function is below a given threshold. This technique should adapt better to a given problem, but it requires knowledge about the problem in order to construct the merit function. For constraint satisfaction problems, the penalty function is a suitable merit function [Nocedal and Wright, 1999].

In this paper, we study variants of the first two termination criteria (based on iteration limits) and leave the others for future work. We experiment with different combinations of ASA, fminsearch (Nelder-Mead algorithm), fmincon (SQP algorithm), and LOQO. Since global search methods have different exploration and convergence properties, more than two of them can also be cascaded in a search process. For example, ASA is powerful in exploring complex global spaces, whereas fminsearch is good at search spaces with relatively simple global structures. Hence, we also experiment with the combinations ASA+fminsearch+fmincon and ASA+fminsearch+LOQO, which allow ASA to explore globally to find promising local regions, fminsearch to exploit structures of the local area while overcoming small local minima, and the local solver to converge to a feasible solution.

4 Test Problems

Since our performance analysis is experimental in nature, an important part of the analysis is choosing appropriate problem ensembles for testing algorithms, as is done in the study of discrete problems [Hogg *et al.*, 1996b; Michalewicz and Schoenauer, 1996]. Regarding continuous constraint satisfaction, there is an unfulfilled need for generic benchmark problem sets, in spite of the work in the mathematical programming and evolutionary computing communities. In the mathematical programming field, test problem sets such as CUTE (Constrained and Unconstrained Testing Environment) [CUTE, 2001] and COPS (Constrained Optimization ProblemS) [Bondarenko *et al.*, 1998] provide very limited ability to generate problems with different characteristics by changing problem parameters. In the evolutionary computing field, a flexible test-case generator proposed recently [Michalewicz *et al.*, 2000] has the major limitation that it defines a landscape that consists of a collection of optimization functions that are only piece-wise continuous and are defined on different subspaces of equal size. Because derivatives do not exist on the boundary of two subspaces, optimization methods requiring continuous derivatives cannot be applied to these problems.

To study constraint problems of different sizes with various numbers and types of constraints, we need a scalable test suite, in which many of the problem parameters can be changed easily. It should allow one to easily compare different optimization techniques developed in different fields. It should be use-

ful in investigating the various characteristics (e.g., constraint ratio) and complexity of constraint problems, as well as the computational complexity of various solving techniques.

In this section, we present two flexible test-case generators for continuous constraint satisfaction problems (CSPs). Both are based on trigonometric functions and generate constraint problems similar to those appearing in robot control applications.

4.1 Sum-form Continuous CSPs

The first test-case generator was first proposed in [Fromherz *et al.*, 2001], which borrows ideas from randomly generated SAT problems. It generates constraints using either a fixed-clause-length model or a constant-density model [Selman *et al.*, 1996]. In the fixed-clause-length model, each constraint has a fixed number of variables that are randomly chosen. In contrast, in the constant-density model each variable may appear in a constraint with some equal probability.

The parameters of the generator $G1(n, m, k, p, t)$ are the number of variables (n), the number of constraints (m), the maximum number of variables in a constraint (k), the probability that a variable is in a clause (p), and the average tightness of the constraints ($t, 1 \geq t \geq 0$). All the parameters of the generator can be changed independently. For instance, $G1(n, m, 3, 1, t)$ generates constraints all containing 3 variables, whereas $G1(n, m, n, 3/n, t)$ generates constraints with a variable number of variables, ranging from 1 to n , with an average of 3.

The tightness t of a (continuous) constraint is defined as the ratio of the size of the space made infeasible by the constraint to the size of the total problem space. For example, $t = 0$ means that the constraint leaves the entire problem space feasible, $t = 1/2$ means that on average it makes half of the problem space infeasible, and $t = 1$ means that it makes the whole problem space infeasible.

One implementation of the generator is to generate constraints based on the same base function, but with randomly generated coefficients. For example, the base form of $G1(n, m, k, 1, t)$ proposed in [Fromherz *et al.*, 2001] is

$$\frac{1}{\sum_{i=1}^k a_i} \sum_{i=1}^k a_i \sin(2\pi\alpha b_i x_{j_i} + 2\pi c_i) \leq \theta \quad (1)$$

where $\{j_1, \dots, j_k\} \subseteq \{1, \dots, n\}$ are the indices of k randomly chosen variables. $0 \leq a_i, c_i \leq 1$ and $1 \leq b_i \leq 2$ are random coefficients. a_i specifies the magnitude of the sine function, b_i its frequency, and c_i its phase. α controls the maximum frequency. A larger value of α usually makes the feasible region specified by the constraint more disjoint. $\alpha = 1$ is used in our experiments. $-1 \leq \theta \leq 1$ specifies the threshold for satisfying the constraint, which is related to the tightness t . The larger its value, the easier the constraint is to satisfy. The θ value to achieve the average tightness specified by t is found experimentally. For $\theta = 0$, on average half of the search space is feasible for the constraint. The left hand side of Eq. (1) are normalized to have values between -1 and 1.

Figure 2 shows the function surfaces of four random constraints generated by $G1(2, 4, 2, 1, 1/4)$ in the form of Eq. (1) and the 2-D feasible space formed by the combination of the four random constraints.

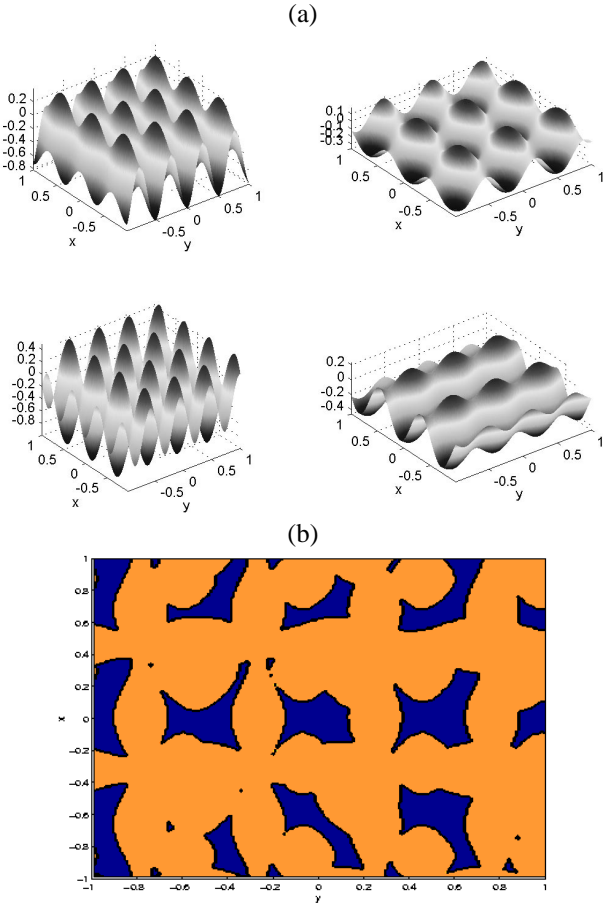


Figure 2: The top figure (a) shows 3-D plots of four random constraints generated by $G1(2, 4, 2, 1, 1/4)$. The bottom figure (b) shows the 2-D feasible spaces (dark regions) formed by the four random constraints.

4.2 Sum-of-product-form Continuous CSPs

The second generator involves sums of products of sine and cosine components, which commonly appear in robot control problems. The base function of the constraints is

$$\frac{1}{\sum_{i=1}^{NS} \prod_{j=1}^{NP_i} c_{ij}} \sum_{i=1}^{NS} \prod_{j=1}^{NP_i} c_{ij} \sin\left(2\pi x_{r_{ij}} + \delta_{ij} \frac{\pi}{2}\right) \leq \theta \quad (2)$$

where $r_{ij} \in \{1, \dots, n\}$ is the index of a randomly chosen variable. $1 \geq c_{ij} \geq -1$ is a random real coefficient specifying the magnitude of the sine function. δ_{ij} is a random coefficient that takes values 0 or 1, which makes the term either a sine function or a cosine function. NS , a positive integer, is the number of sum terms, and NP_i , a positive integer, is the number of product terms in each sum term. $-1 \leq \theta \leq 1$ specifies the threshold for satisfying the constraint, which again is related to the tightness parameter t . The left hand side of Eq. (2) is normalized to have values between -1 and 1.

Based on constraints defined in Eq. (2), the parameters of the second generator $G2(n, m, k_s, k_p, p_s, p_p, t)$ are the number of variables (n), the number of constraints (m), the maximum number of sum terms in a constraint ($k_s, k_p \geq 1$), the

maximum number of product terms in any sum term of a constraint ($k_p, k_p \geq 1$), the probability determining the number of sum terms in a constraint ($p_s, 1 \geq p_s \geq 0$), the probability determining the number of product terms in a sum term ($p_p, 1 \geq p_p \geq 0$), and the average tightness of the constraints ($t, 1 \geq t \geq 0$).

All the parameters of the generator can be changed independently. Constraints are generated with randomly generated coefficients. By changing the tightness parameter t in $G2(n, m, k_s, k_p, p_s, p_p, t)$, we can control the difficulty of the constraints, which in turn affects the complexity of the whole problem.

This generator can also generate constraints in either the fixed-clause-length or constant-density model. By setting p_s and p_p to 1, $G2(n, m, k_s, k_p, 1, 1, t)$ generates constraints with a fixed number ($k_s k_p$) of sine elements, each containing one variable. For example, the constraint generated by $G2(n, m, 3, 1, 1, 1, t)$ consists of a sum of three sine terms. Similarly, the constraint generated by $G2(n, m, 1, 3, 1, 1, t)$ consists of a product of three sine elements.

On the other hand, when p_s or p_p is not 1, the constraints follow the constant-density model. For example, $G2(n, m, n, 1, 3/n, 1, t)$ generates constraints with different numbers of sine terms, ranging from 1 to n (with one sine element per term), with an average of 3 terms.

5 Experimental Results

In this section, we show the improvement of cooperative solvers in solving continuous CSPs. The test problems are generated using test-case generators G1 and G2. In our experiments, to guarantee a random CSP generated by G1 or G2 is satisfiable, we make sure that the origin is a feasible point of all its constraints. When we generate a constraint for which the origin is not a feasible point, the constraint is simply discarded.

In solving nonlinear constraint problems, some solvers accept problems in the constrained form, whereas others only work with a single unconstrained objective function. Among our solvers, only `fmincon` takes the constrained form. For the other solvers, we convert a constraint problem into an unconstrained penalty function problem [Nocedal and Wright, 1999].

Multi-start is used with all the solvers in order to solve the test problems. Starting from an initial point, these solvers cannot guarantee to find a feasible solution for the nonlinear problems. Hence, in the experiments, we keep restarting a solver from different starting points until it finds a feasible solution. The initial and restart points of solvers `fmincon`, `fminunc`, `LOQO`, and `fminsearch` are generated with the following pre-sampling scheme: n random points are sampled within a fixed bounding box, e.g., $[-1, 1]^n$, and the best of them is chosen as the initial or restart point. The best sample point is the one with the smallest constraint violation, which may be close to or even inside a feasible region. In our previous work, we found that the pre-sampling scheme improved the solvers' performance over just picking a single random point from the fixed bounding box [Fromherz *et al.*, 2001; Shang *et al.*, 2001]. As an exception, ASA does not use the pre-sampling scheme. Instead, a random point within the

fixed bounding box $[-1, 1]^n$ is used as the initial or restart point. The reason for not doing pre-sampling is that at high temperatures ASA jumps quite freely across the search space. A relatively good starting point produced by the pre-sampling scheme does not help ASA as much as the other solvers.

Since the constraint ratio has been identified as a key indicator of complexity in discrete CSPs, in this paper we compare the performance of various solvers based on continuous CSPs with varying constraint ratios. Given a constraint ratio, 100 random instances are generated and the performance of the solvers is compared based on the median numbers of function evaluations for solving the instances. We use medians instead of means because the means of the number of function evaluations are heavily influenced by a small number of extremely large values. Our interest is in what the majority of instances from the distribution are like, not in the unusual or extreme cases. As the median is more robust in the presence of such outliers, it appears to be a more informative statistic for our current purpose.

5.1 Results for Sum-form Continuous CSPs

This set of experiments is based on test problems generated by $G1(n, m, 3, 1, 1/4)$ with $n = 25$ and 50 and varying m . We first compare the performance of individual solvers, and then show the improvement of several cooperative solvers. The individual solvers we tried are fmincon, fminunc, LOQO, and ASA. Each run of ASA is limited to 1000 iterations, while the other solvers terminate when the improvement is less than their default parameters. The cooperative solvers we tried are fminsearch+fmincon, ASA+fmincon, ASA+LOQO, ASA+fminsearch+fmincon, ASA(L)+fminsearch+fmincon, and ASA+fminsearch+LOQO. In each run of fminsearch+fmincon, fminsearch runs for 300 iterations from a given starting point. If fminsearch finds a solution, this run is finished. Otherwise, the best point found by fminsearch is used as the starting point for fmincon. In ASA+fmincon and ASA+LOQO, ASA is run for 1000 iterations. In ASA+fminsearch+fmincon and ASA+fminsearch+LOQO, ASA is run for 700 iterations, then fminsearch for 300 iterations, followed by fmincon or LOQO. Finally, in ASA(L)+fminsearch+fmincon, the maximum iteration limit (L) of ASA is a linear function of the constraint ratio r when r is larger than 3: $L = \max(1, 500(r - 3))$.

Figure 3 compares the performance of selected individual solvers. The figure shows that LOQO consistently outperforms the other solvers for various constraint ratios. No clear phase transition behavior is found in its complexity curve. The complexity curve of fminunc has a similar shape to that of LOQO, and is worse than LOQO by a constant factor. fmincon has a quite different complexity curve. It works well for weakly constrained problems, i.e., constraint ratio less than 3, but becomes much worse for highly constrained problems. There seems to be a phase transition from easy to hard around the ratio of 3 for fmincon. ASA performs the worst among these four solvers, and its complexity increases quickly as the ratio increases. fminsearch is even worse than ASA, and its data is not shown in the figure.

The improvement with cooperative solving is shown in Figure 4. Although fmincon, fminsearch, and ASA work

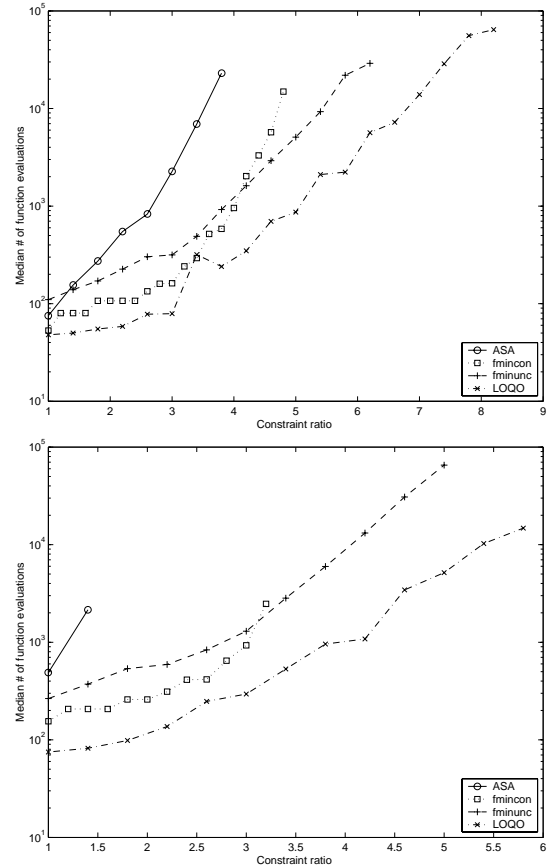


Figure 3: Performance comparison of individual solvers fmincon, fminunc, LOQO, and ASA on continuous sum-form CSPs generated by $G1(n, m, 3, 1, 1/4)$ with $n = 25$ (top diagram) and 50 (bottom diagram).

poorly on moderately constrained problems individually, a cooperative combination of global and local search has significantly improved performance. Compared to fmincon, although fminsearch+fmincon is more expensive for small constraint ratios, its cost increases at a slower rate than that of fmincon and eventually becomes better for large constraint ratios. The complexity curve for fminsearch+fmincon has phases increasing-flat-increasing, which can be explained by fminsearch's overhead. In the first phase where the constraint ratio is small and the problems are easy, fminsearch usually finds a solution within 300 iterations, within which it needs more iterations as the constraint ratio increases. In the second phase, where the constraint ratio is moderate, fminsearch is not able to find a solution within 300 iterations. However, a single fminsearch run usually is sufficient to find a good starting point for fmincon to find a solution. That is why the curve is nearly flat in this phase. Finally, in the third phase, where the constraint ratio is large, multiple restarts of fminsearch+fmincon are required.

The cooperative solvers ASA+fminsearch+fmincon and ASA(L)+fminsearch+fmincon use a more powerful global search method than fminsearch+fmincon and further improve the performance on highly constrained problems. fminsearch

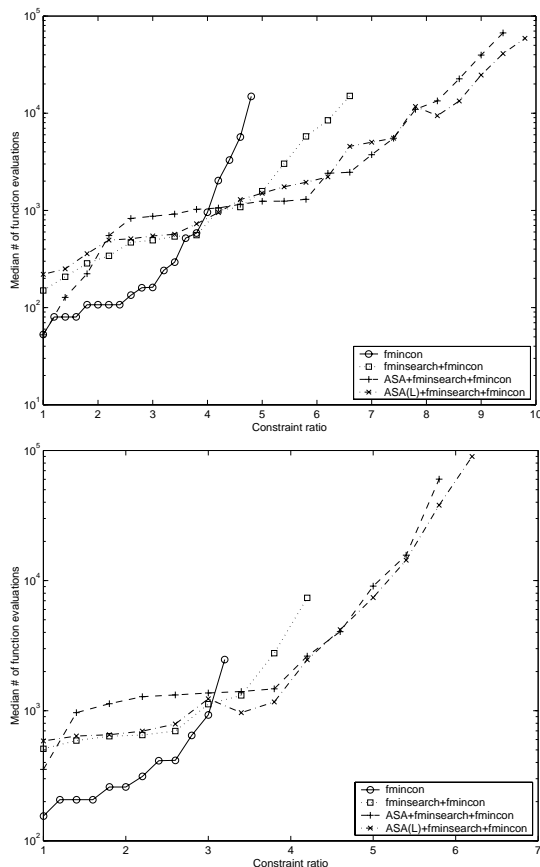


Figure 4: Performance improvement of cooperative solvers fminsearch+fmincon and ASA+fminsearch+fmincon on highly constrained problems generated by $G1(n, m, 3, 1, 1/4)$ with $n = 25$ (top diagram) and 50 (bottom diagram).

does a limited form of global search, whereas ASA is a true global search method. ASA+fminsearch+fmincon and ASA(L)+fminsearch+fmincon combine a full-blown global search, limited global search, and local search. Figure 4 shows that they are very effective on harder, highly constrained problems. In ASA(L)+fminsearch+fmincon, only one iteration of ASA is run when the constraint ratio is less than or equal to 3. That is why it has a complexity curve similar to that of fminsearch+fmincon in this region. When the constraint ratio is more than 3, the iteration limit of ASA increases proportionally to the constraint ratio, which leads to better results than the fixed iteration limit of ASA in ASA+fminsearch+fmincon. As with fminsearch+fmincon, the two solvers have phases increasing-flat-increasing, which can be explained by ASA and fminsearch's overheads.

Comparing the solvers shown in Figure 4, the local solver fmincon is the best method for weakly constrained problems, in which feasible solutions are plentiful. When problems become harder, more global search is beneficial. The right amount of global search depends on the difficulty of the search space. When the search space is moderately rugged, a limited global search performed by fminsearch may be sufficient. However, when the space is very hard, more extensive global

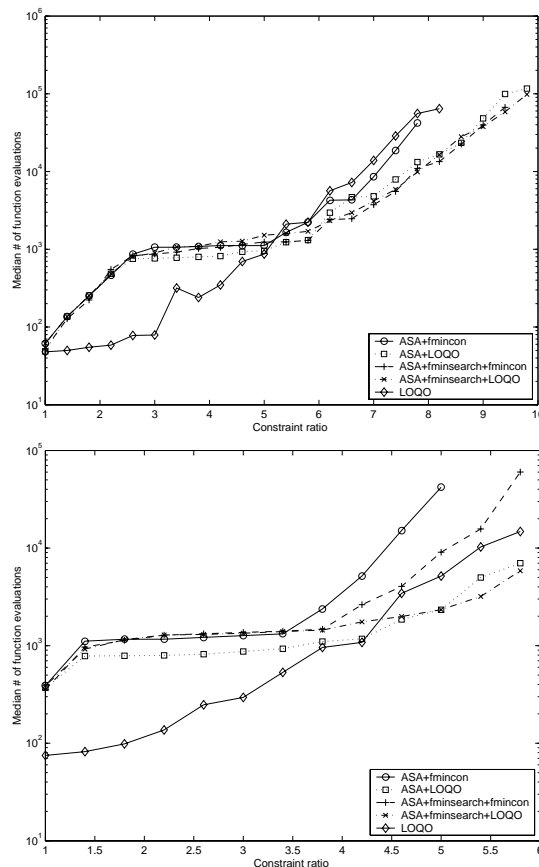


Figure 5: Performance comparison of four combined solvers involving ASA on continuous sum-form CSPs generated by $G1(n, m, 3, 1, 1/4)$ with $n = 25$ (top diagram) and 50 (bottom diagram).

search as carried out by ASA is better.

Better components in a cooperative solver make the solver better. Figure 5 shows how the performance of a cooperative solver improves when fmincon is replaced by LOQO. The four solvers compared in the figure are ASA+fmincon, ASA+LOQO, ASA+fminsearch+fmincon, and ASA+fminsearch+LOQO. The two solvers using LOQO are better than the two using fmincon because LOQO has a better success rate than fmincon, given the starting points provided by ASA or ASA+fminsearch. The complexity curves of these solvers have similar phases (again increasing-flat-increasing). Note that these solvers all perform better than the individual solvers shown in Figure 3 for highly constrained problems.

Local solvers need very good starting points, especially for highly constrained problems, in order to have a good chance of finding solutions, whereas cooperative solvers are more resilient. Figure 6 plots the success rates of the fminsearch+LOQO portion and the LOQO portion of solver ASA+fminsearch+LOQO. 100 random instances generated by $G1(25, m, 3, 1, 1/4)$ are solved for each constraint ratio. Lighter areas correspond to higher success rates. The top diagram shows the success rates of the fminsearch+LOQO portion plotted against varying constraint ratios and the penalty

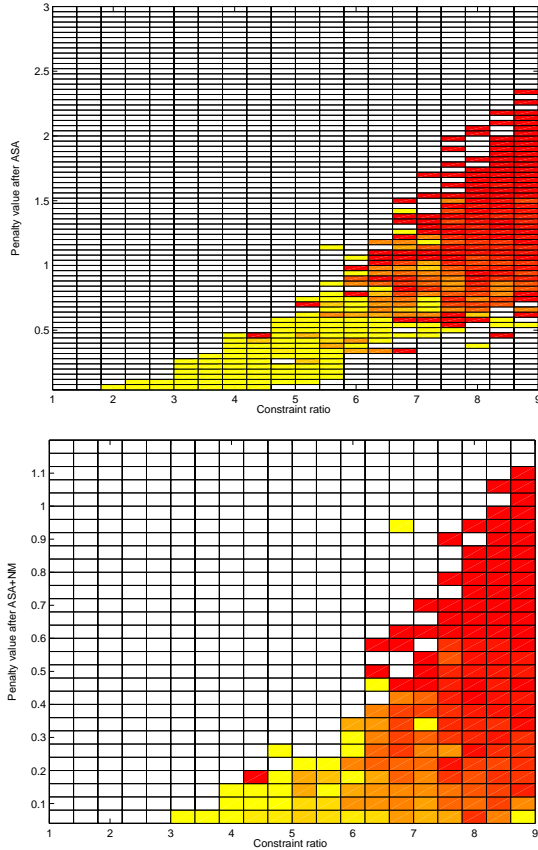


Figure 6: The success rates of the fminsearch+LOQO portion (top diagram) and the LOQO portion (bottom diagram), respectively, of solver ASA+fminsearch+LOQO against the penalty values of their starting points. 100 random instances of sine-based sum-form CSPs generated by $G1(25, m, 3, 1, 1/4)$ are solved for each constraint ratio.

values of the starting points as provided by ASA. (Runs in which ASA found feasible solutions are not included.) The penalty values of the starting points increase as the constraint ratio increases, meaning the points found by ASA are getting worse. The diagram also shows that, to have a good chance to succeed, fminsearch+LOQO needs to start from points with small penalty values, e.g., less than 1. For example, when the constraint ratio is less than or equal to 4, the penalty values of points found by ASA are in the range of 0 to 0.5, and starting from these points fminsearch+LOQO was always able to find feasible solutions.

The bottom diagram shows the success rates of the LOQO portion, again plotted against varying constraint ratios and the penalty values of the best points found by ASA+fminsearch. There are fewer data points in this diagram than in the top one because some runs of fminsearch found feasible solutions. The diagram shows that, to have a good chance to succeed, LOQO needs to start from points with even smaller penalty values, e.g., less than 0.3. In addition, as the constraint ratio increase, LOQO needs to have increasingly better starting points to be successful.

5.2 Results for Sum-of-product-form Continuous CSPs

In this section, we present results for CSPs generated by $G2(n, m, k_s, k_p, p_s, p_p, t)$, mainly to confirm the observations made so far. We show the performance improvement of cooperative solvers ASA+fmincon, fminsearch+fmincon, and ASA+fminsearch+fmincon over local solver fmincon. The iteration limits of fminsearch and ASA in these cooperative solvers are selected after a few trials. They are good enough to illustrate the general behaviors of the cooperative solvers, and are likely not the optimal values for the solvers. fminsearch is limited to 300 iterations in fminsearch+fmincon and ASA+fminsearch+fmincon. ASA is limited to 1000 iterations in ASA+fmincon and 700 iterations in ASA+fminsearch+fmincon.

Figure 7 shows the results for problems generated by $G2(n, m, 3, 3, 1, 1, 1/4)$ with $n = 25$ and 50, whereas Figure 8 shows those for problems generated by $G2(n, m, 3, 3, 3/n, 3/n, 1/4)$. 100 random instances were tried for each constraint ratio. The solvers have similar behaviors on the two types of CSPs. Similar to the results on CSPs generated by $G1$, the cooperative solvers are better for more constrained problems, whereas local solver fmincon is good for weakly constrained problems. For highly constrained problems, ASA+fminsearch+fmincon is always the best. However, fminsearch+fmincon and ASA+fmincon switch places on the two types of problems. fminsearch+fmincon is better than ASA+fmincon on $G2(n, m, 3, 3, 1, 1, 1/4)$ problems, but worse on $G2(n, m, 3, 3, 3/n, 3/n, 1/4)$ problems.

6 Conclusions

In this paper, we study cooperative solvers for continuous constraint problems, and focus on the approach of combining global and local search in series. In this approach, the global search method explores the global structure of the search space and identifies promising sub-regions, providing good starting points to local search methods. In our experiments, we demonstrate that cooperative solvers of this nature can achieve significant improvement on hard, highly constrained problems, even though the local and global solvers alone work poorly.

To optimize the speed and quality of constraint solving, we need to balance the global and local search. A general guideline is that more global search is necessary as the problems become more constrained. The extent of global search is affected by the type of the global solver, as well as budgets such as the time limit allocated to it. Global search methods have different abilities to explore a global search space. For example, simulated annealing is powerful in exploring an arbitrary, complex search space, whereas Nelder-Mead is limited to following relatively simple global structures. To take advantage of the strengths of different global methods, we may combine two or more of them in the cooperative solvers.

After selecting the global and local solvers, determining when to switch from one to another is non-trivial. The decision may be static, such as based on a fixed number of iteration of the solvers, or adaptive according to the online progress of

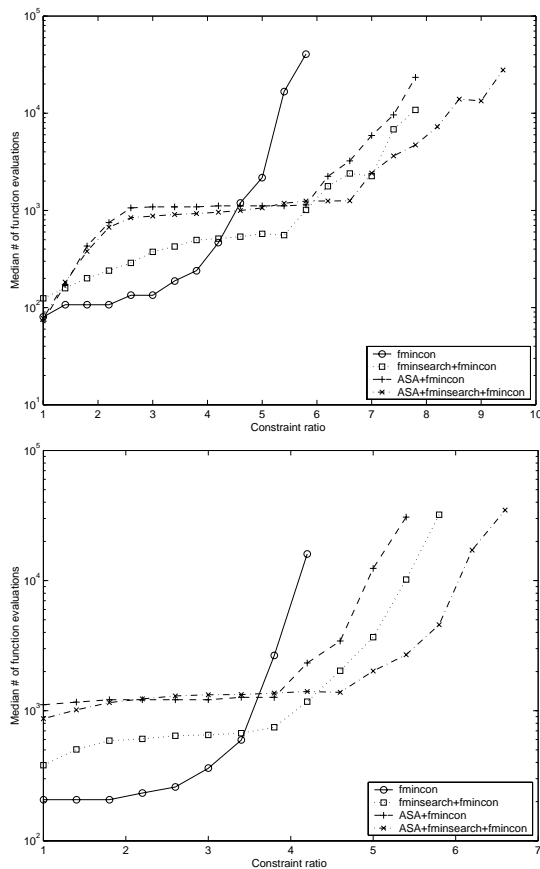


Figure 7: Performance comparison of individual and combined solvers on continuous sum-of-product-form CSPs generated by $G2(n, m, 3, 3, 1, 1, 1/4)$ with $n = 25$ (top diagram) or 50 (bottom diagram).

the solvers. In this paper, we study variants of the static approach based on fixed iteration limits and a pre-determined relation of iteration limits and constraint ratios.

As mentioned, there is strong indication that the balance between global and local solving should be adaptive to problem characteristics such as the number of variables and the constraint ratio. More work is necessary to study additional key problem parameters, such as the tightness of constraints, correlations of constraints, ruggedness of search space, etc., for better characterizing the complexity of constraint problems and solution methods. Based on this work, we plan to study adaptive cooperative solving approaches in the future.

References

- [Bondarenko *et al.*, 1998] A. S. Bondarenko, D. M. Bortz, and J. J. Moré. COPS: Large-scale nonlinearly constrained optimization problems. Technical Memorandum ANL/MCS-TM-237, Argonne National Laboratory, Argonne, Illinois, 1998.
- [CUTE, 2001] CUTE. Constrained and unconstrained testing environment, 2001. <http://www.cse.clrc.ac.uk/Activity/CUTE>.

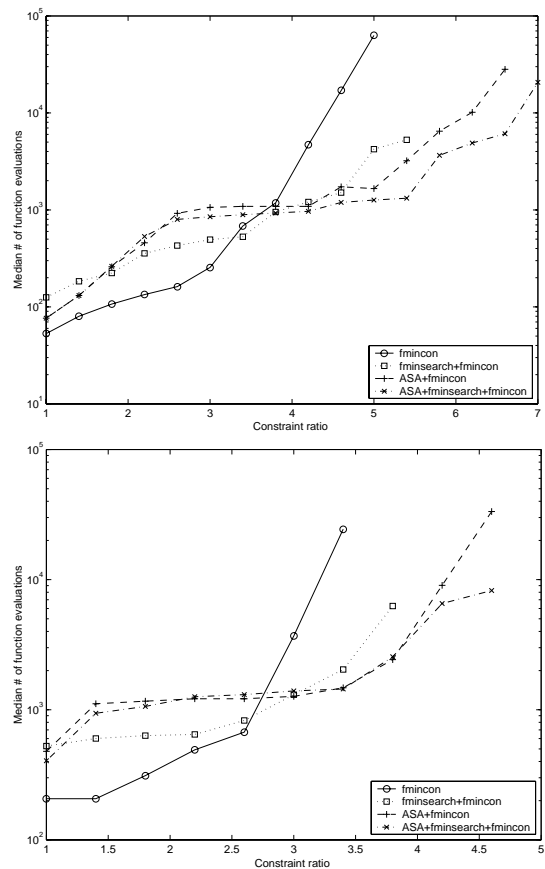


Figure 8: Performance comparison of individual and combined solvers on continuous sum-of-product-form CSPs generated by $G2(n, m, 3, 3, 3/n, 3/n, 1/4)$ with $n = 25$ (top diagram) or 50 (bottom diagram).

- [Dolan and Moré, 2001] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. Technical Memorandum ANL/MCS-P861-1200, Argonne National Laboratory, Argonne, Illinois, 2001.
- [Fourer *et al.*, 1993] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Boyd & fraser publishing company, 1993.
- [Fromherz *et al.*, 2001] M. P. J. Fromherz, T. Hogg, Y. Shang, and W. B. Jackson. Modular robot control and continuous constraint satisfaction. In *Proc. IJCAI-01 Workshop on Modelling and Solving Problems with Constraints*, pages 47–56, Seattle, WA, 2001.
- [Hart, 1994] W. E. Hart. *Adaptive global optimization with local search*. PhD thesis, University of California, San Diego, 1994.
- [Hogg *et al.*, 1996a] T. Hogg, B. A. Huberman, and C. P. Williams, editors. *Artificial Intelligence, Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81:1-2. Elsevier, March 1996.

- [Hogg *et al.*, 1996b] Tad Hogg, Bernardo A. Huberman, and Colin Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.
- [Ingber, 2001] L. Ingber. Adaptive simulated annealing, 2001. <http://www.ingber.com/>.
- [Kitts, 1997] B. Kitts. Regulation of complex systems. In *Proc. 1st Int'l Conf. on Complex Systems*, September 1997.
- [Lagarias *et al.*, 1998] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the Nelder-Mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.
- [Martin, 1996] O. C. Martin. Combining simulated annealing with local search heuristics. In G. Laporte and I. Osman, editors, *Metaheuristics in Combinatorial Optimization*, pages 57–75. Annals of Operations Research Vol. 63, 1996.
- [McKinnon, 1998] K. I. M. McKinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.
- [Michalewicz and Schoenauer, 1996] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.
- [Michalewicz *et al.*, 2000] Z. Michalewicz, K. Deb, M. Schmidt, and T. Stidsen. Test-case generator for constrained parameter optimization techniques. *IEEE Transactions on Evolutionary Computation*, 4(3):197–215, September 2000.
- [Muhlenhein *et al.*, 1988] H. Muhlenhein, M. Georges-Schleuter, and O. Kramer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(65), 1988.
- [Murphy and Baker, 1995] M. J. Murphy and E. L. Baker. GLO: Global local optimizer. LLNL unclassified code #960007, Lawrence Livermore National Laboratory, Livermore, CA, November 1995. See also <http://www.llnl.gov/glo>.
- [Nelder and Mead, 1965] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.
- [Nocedal and Wright, 1999] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.
- [Pinter, 1996] J. D. Pinter. *Global Optimization in Action*. Kluwer Academic Publishers, 1996.
- [Powell, 1983] M. J. D. Powell. Variable metric methods for constrained optimization. In A. Bachem, M. Grottschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 288–311. Springer-Verlag, 1983.
- [Sakata, 2001] S. Sakata. ASAMIN - a MATLAB gateway routine to adaptive simulated annealing software, 2001. <http://www.econ.lsa.umich.edu/~sakata/software/>.
- [Selman *et al.*, 1992] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, San Jose, CA, 1992.
- [Selman *et al.*, 1996] Bart Selman, David Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81:17–29, 1996.
- [Shang *et al.*, 2001] Y. Shang, M. P. J. Fromherz, T. Hogg, and W. B. Jackson. Complexity of continuous, 3-SAT-like constraint satisfaction problems. In *Proc. IJCAI-01 Workshop on Stochastic Search Algorithms*, pages 49–54, Seattle, WA, 2001.
- [Vanderbei and Shanno, 1999] R. J. Vanderbei and D. F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Comp. Optim. Appl.*, 13:231–252, 1999.
- [Yokoo, 1997] M. Yokoo. Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analyzing Landscapes of CSPs. In *Proc. of CP'97, number 1330 in LNCS*, pages 357–370. Springer-Verlag, 1997.