Complexity of Continuous, 3-SAT-like Constraint Satisfaction Problems

Yi Shang, Markus P.J. Fromherz, Tad Hogg
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{yshang,fromherz,hogg}@parc.xerox.com

Warren B. Jackson
HP Labs
1501 Page Mill Road, MS 4U-12
Palo Alto, CA 94304
warren_jackson@hp.com

Abstract

Continuous constrained optimization is at the core of many real-world applications such as planning, scheduling, control, and diagnosis of physical systems (car, planes, factories). 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, we analyze continuous constraint satisfaction problem formulations based on (discrete) 3-SAT problems, which have a strong relation between structure and search cost. We compare the search complexities of three different problem formulations and three randomized search algorithms. This allows us not only to compare different problems and solution approaches, but also to connect back to results from similar studies on SAT problems. In particular, we find that median search cost is characterized by simple parameters such as the constraint-to-variable ratio, and that discrete search algorithms such as GSAT have continuous counter-parts with similar behavior.

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, because either the underlying physical system is continuous or because discrete implementations exhibit abrupt changes in outputs for small changes in inputs. Such manifest nonlinearities often lead to undesirable behaviors – limit cycles, chaos, and oscillations – when such systems interact with the real world [Strogatz, 1994]. Continuous constrained optimization minimizes these problems. 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 some of the progress made in understanding complexity results for discrete optimization problems to continuous problems.

In particular, significant progress has been made in understanding problem complexity of the (discrete) satisfiability (SAT) problem [Hogg *et al.*, 1996a]. The SAT problem belongs to an important class of discrete constraint-satisfaction problems (CSP). Many problems in artificial intelligence, logic, computer aided design, database query, and planning, etc. can be formulated as SAT problems.

Generally, a *SAT problem* is defined as follows. Given a set of m clauses $\{C_1, C_2, \cdots, C_m\}$ on n Boolean variables $x = (x_1, x_2, \cdots, x_n)$, $x_i \in \{0, 1\}$, and a Boolean formula in conjunctive normal form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n, \tag{1}$$

find an assignment of values to the variables so that (1) evaluates to be *true*, or derive its infeasibility if (1) is infeasible. In the well-studied 3-SAT problem, clause C_i involves exactly three variables that may appear either positively or negatively, as in $C_i = x_j \vee x_k \vee \neg x_l$ $(j, k, l \in \{1, ..., n\})$.

SAT problems can be formulated as discrete or continuous, constrained or unconstrained, optimization problems. In the past, constrained programming algorithms have been developed to solve SAT problems transformed into integer programming problems. In a typical example [Kamath *et al.*, 1990], a SAT clause C_i is transformed into a linear inequality:

$$\sum_{j=1}^{n} q_i(y_j) \geq 2 - |C_i| \tag{2}$$

where $y_j \in \{-1, 1\}$ is the jth variable and

$$q_i(y_j) = \begin{cases} y_j & \text{if literal } x_j \text{ is in clause } C_i \\ -y_j & \text{if literal } \neg x_j \text{ is in clause } C_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

While the simplex method is effective for solving linear programming problems, integer linear programming problems are much harder to solve. Various methods such as branch-and-bound, cutting plane [Hooker, 1988], and interior point methods [Kamath *et al.*, 1990] have been applied to solve SAT problems. Previous results show that integer programming methods may be faster than resolution for certain classes of problems, although they often fail to solve hard SAT instances [Gu *et al.*, 1997].

In formulating a continuous SAT problem, discrete variables in the original problem are transformed into continuous

variables in such a way that solutions to the continuous problem are solutions to the original problem. The continuous formulation is potentially beneficial because the continuous function can provide a continuous path from a given infeasible point to a feasible one without using discontinuous jumps. Another motivation for using a continuous formulation is that it allows the application to employ continuous reasoning similar to fuzzy logic. A disadvantage is that continuous formulations often require computationally expensive algorithms [Gu *et al.*, 1997].

Some CSPs are NP-complete and require algorithms of exponential complexity in the worst case to obtain a satisfying assignment. One particularly well-studied phenomenon for discrete SAT problem formulations is the *complexity phase transition* [Hogg *et al.*, 1996b]. For a variety of complete search algorithms, 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 underconstrained) as well as when this ratio is large (where the problem is overconstrained), while solving time is largest in the intermediate case. For satisfiable problems, incomplete algorithms such as GSAT [Selman *et al.*, 1992] exhibit similar complexity curves [Yokoo, 1997]. This paper compares these results with a study of continuous problem formulations and (incomplete) randomized continuous constraint solving algorithms.

The rest of this paper is structured as follows. In the next section, we present three different continuous problem formulations of the 3-SAT problem. In Section 3, the problems are solved and analyzed using three variations of a randomized continuous constraint solving algorithm. As for discrete 3-SAT studies, measures of the problem solving difficulty are examined as a function of the ratio of constraints to variables. The solution complexity semi-quantitatively follows the behavior exhibited by discrete problems and exhibits a distinct change in behavior as the ratio of constrains to variables reaches 2.0. Section 4 discusses these results and compares them to results from discrete formulations. Section 5 closes with conclusions and an outline of future work.

2 Problem Formulations

We present three alternative formulations for continuous constraint satisfaction problems. Based on the concept of clauses in 3-SAT, constraints in the three formulations are represented by quadratic, sigmoid, and exponential functions. The common characteristics of these continuous formulations are as follows.

- Each positive literal x in a 3-SAT clause is converted to a continuous function $f(x)$, and each negative literal $\neg x$ is converted to $f(1-x)$, where f depends on the formulation.
- Disjunction (e.g., $x_1 \vee x_2$) is represented as addition in the constraint (e.g., $f(x_1) + f(x_2)$).
- Each clause (e.g., $x_1 \vee x_2 \vee \neg x_3$) is transformed into an inequality constraint (e.g., $f(x_1)+f(x_2)+f(1-x_3) \geq c$, where constant c depends on the formulation).

In the rest of this section, we illustrate the continuous formulations by using a simple example, the 3-SAT clause $x_1 \vee x_2 \vee \neg x_3$.

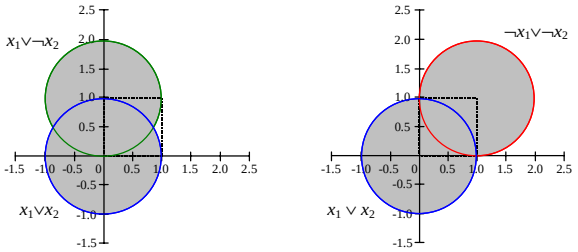

Figure 1: 2-D feasible spaces of the quadratic formulation for two 2-SAT problems (white regions).

2.1 Quadratic formulation

The basic transformation function of the quadratic formulation is

$$f(x) = x^2 \tag{4}$$

and the constant limit for the inequality is $c = 1$. Based on this function, clause $x_1 \vee x_2 \vee \neg x_3$ is transformed into the quadratic constraint

$$x_1^2 + x_2^2 + (1 - x_3)^2 \geq 1 \tag{5}$$

Using this transformation, a 3-SAT problem with n Boolean variables and m clauses is converted into a CSP with n continuous variables and m continuous constraints. A feasible solution to the SAT clause, i.e., $x_1 = 1$, $x_2 = 1$, or $x_3 = 0$, always satisfies the corresponding continuous constraint.

As an illustration, Figure 1 shows 2-D feasible spaces of continuous CSPs created from two 2-SAT problems. The left diagram is for the SAT problem $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, and the right diagram is for the SAT problem $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$. The interior of the dark areas are infeasible regions, whereas the white areas including points on the boundary are feasible regions.

Solutions to the continuous problems can be mapped to solutions to the original (discrete) SAT problems through the following thresholding: if a value x_i of the continuous solution is equal to or larger than 0.5, then set x_i to 1 in the solution to the SAT problem; otherwise set x_i to 0. Note that a mapping like this does not guarantee that a solution to the continuous problem is always mapped to a feasible solution to the original SAT problem. For example, considering the right diagram in Figure 1, when both x_1 and x_2 have large positive or large negative values at the same time, they are feasible solutions to the continuous CSP. However, after thresholding, they are not feasible solution to the original SAT problem.

This issue can be addressed by introducing simple bounds on the variables in the continuous formulation. For example, by adding the constraints

$$-0.5 \leq x_i \leq 1.5 \quad \text{for} \quad i = 1, \ldots, n, \tag{6}$$

a solution to the continuous problem is always mapped to a feasible solution to the SAT problem using the 0.5 threshold.

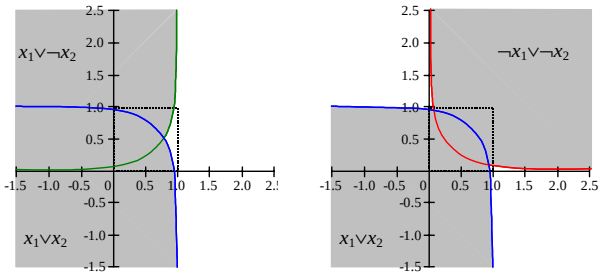

Figure 2: 2-D feasible spaces of the sigmoid formulation for two 2-SAT problems (white regions).

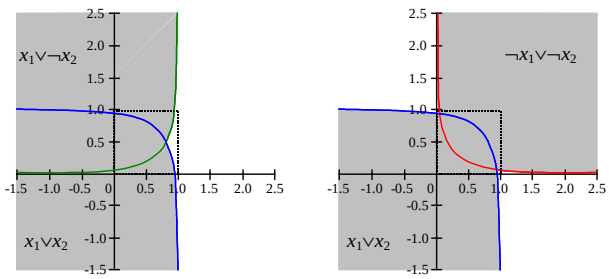

Figure 3: 2-D feasible spaces of the exponential formulation for two 2-SAT problems (white regions).

2.2 Sigmoid formulation

The basic transformation function of the sigmoid formulation is

$$f(x) = \frac{1}{1 + e^{-\alpha(x-1)}} \tag{7}$$

and the constant limit for the inequality is $c = \frac{1}{2}$. Based on this function, clause $x_1 \vee x_2 \vee \neg x_3$ is transformed into the nonlinear constraint

$$\frac{1}{1 + e^{-\alpha(x_1-1)}} + \frac{1}{1 + e^{-\alpha(x_2-1)}} + \frac{1}{1 + e^{\alpha x_3}} \geq \frac{1}{2} \tag{8}$$

The constant α is chosen to make sure that a solution to the continuous constraint can be mapped to a feasible solution to the original problem through thresholding. In other words, the center of the search space between 0 and 1, the vector (0.5, 0.5, $\cdots$, 0.5), should not satisfy the continuous constraint. For 3-SAT problems, we have

$$\frac{3}{1 + e^{0.5\alpha}} < \frac{1}{2} \qquad \Longrightarrow \qquad \alpha > 2 \ln 5 \approx 3.2189$$

We arbitrarily set $\alpha = 3.3$ in our experiments.

Using this formulation, a feasible solution to a SAT problem, such as $x_1 = 1$, $x_2 = 1$, or $x_3 = 0$ for the sample SAT clause, is always a feasible solution to the continuous CSP, and vice versa. A solution to the continuous CSP is always mapped to a feasible solution to the original SAT problem through simple thresholding: set x_i to 1 if $x_i \geq 0.5$; otherwise set x_i to 0. Recall that the unbounded quadratic formulation does not have this property.

Figure 2 shows 2-D feasible spaces of continuous CSPs created from the same two 2-SAT problems as before, using the sigmoid formulation. The left diagram is for the SAT problem $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, and the right diagram is for the SAT problem $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$. The interior of the dark areas are infeasible regions, whereas the white areas including points on the boundary are feasible regions.

2.3 Exponential formulation

The basic transformation function of the exponential formulation is

$$f(x) = 2^{\beta(x-1)} \tag{9}$$

and the constant limit for the inequality is $c = 1$. Based on this function, clause $x_1 \vee x_2 \vee \neg x_3$ is transformed into the nonlinear constraint

$$2^{\beta(x_1-1)} + 2^{\beta(x_2-1)} + 2^{-\beta x_3} \geq 1 \tag{10}$$

The constant β is chosen to make sure that a solution to the continuous constraint can be mapped to a feasible solution to the original clause through thresholding. In other words, the center of the search space between 0 and 1, (0.5, 0.5, $\cdots$, 0.5), should not satisfy the continuous constraint. For 3-SAT problems, we have

$$3 \times 2^{-0.5\beta} < 1 \qquad \Longrightarrow \qquad \beta > 2 \log_2 3 \approx 3.1699$$

We arbitrarily set $\beta = 3.2$ in our experiments.

This formulation has properties similar to the sigmoid formulation. For example, a feasible solution to a SAT problem, such as $x_1 = 1$, $x_2 = 1$, or $x_3 = 0$ for the sample SAT clause, is always a feasible solution to the continuous CSP, and vice versa. A solution to the continuous CSP is mapped to a feasible solution to the original SAT problem through the same thresholding scheme: set x_i to 1 if $x_i \geq 0.5$ and to 0 otherwise. These two formulations are very similar inside the center region [0,1], but become very different when the variables have large values. In the sigmoid formulation, constraint violation saturates at a constant far away from the center region, whereas in the exponential formulation constraint violation increases to infinity. The latter potentially provides better search guidance to search algorithms when they look for feasible regions.

Figure 3 shows 2-D feasible spaces of continuous CSPs created from the same two 2-SAT problems as before, using the exponential formulation. The left diagram is for the SAT problem $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$, and the right diagram is for the SAT problem $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$. The interior of the dark areas are infeasible regions, whereas the white areas including points on the boundary are feasible regions.

3 Search Algorithms

3.1 Sequential Quadratic Programming and MATLAB implementation

The sequential quadratic programming (SQP) optimization method represents the state-of-the-art in nonlinear programming methods. Based on the work of Biggs, Han, and Powell, the method mimics Newton's method for constrained optimization just as for unconstrained optimization [Powell, 1983]. It is an iterative method starting from some initial point and converges to a constrained local minimum. At each iteration of an SQP method, 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.

The function fmincon is a MATLAB implementation of an SQP method. An fmincon iteration consists of three main stages: (a) updating of the Hessian matrix of the Lagrangian function, (b) quadratic programming problem solution, and (c) line search and merit function calculation. This iteration is repeated until an optimal or feasible solution is found (for optimization or satisfaction problems, respectively). The cost of an iteration is thus on the order of n function evaluations (n the number of variables), as the slope of the Lagrangian function is computed for small step sizes in each of the variables. (A small, constant number of function evaluations is added to the cost for the other two stages of fmincon.)

fmincon is a local search algorithm in the continuous search space, just like GSAT [Selman *et al.*, 1992] is a local search algorithm in the discrete search space. It starts from an initial point and usually converges to a constrained local optimum close to the initial point. fmincon may be trapped by local optima and may not be able to find a feasible solution when constraints are nonlinear. fmincon cannot prove infeasibility when no feasible solution exists.

3.2 Solver 1: Global random restart of local searches

The first algorithm we used is a local search strategy that mimics GSAT by combining fmincon with random global restart. fmincon is started from a random initial point in a certain search region. If it stops without finding a consistent solution, fmincon is restarted from a new random point in the search region. This is repeated until a solution is found. Variable values x_i in the initial and restart points are constrained to be in the interval [0,1].

In working with the quadratic formulation, although the bounded formulation is more closely mapped to the SAT problem, empirically the bounded problem is much harder to solve using fmincon than the unbounded formulation. An intuitive explanation is that feasible regions of the bounded formulation are very small for overconstrained problems, such as when the constraints to variables ratio is over 2. When the constraints are nonlinear and the feasible regions are small, SQP have difficulty in finding feasible solutions.

In using the unbounded formulation to solve SAT problems, we treat the case where the discretized version of a solution to the continuous problem is not a feasible solution to the SAT problem as failure of the local solver. When this happens, we simply restart the continuous local solver from randomly generated points. This is done repeatedly until a feasible solution to the SAT problem is found.

3.3 Solver 2: A simple heuristic in generating better starting points

In our experiments with Solver 1, fmincon often runs only a single iteration between restarts. Our hypotheses is that, as the constraint ratio increases, most restart points are far from feasible regions in the search space, and fmincon immediately gives up. Thus, most runs of fmincon are wasted.

The goal for our second solver was to generate better starting points. Instead of generating a single random point, the algorithm randomly samples k points in the search space and selects the best one as the initial and restart points for fmincon. In our experiments, we set k both to n, the number of variables, and to m, the number of constraints. (Our preliminary observation is that k should probably increase with the constraint ratio.) The best sample point is defined as the point with the smallest constraint violation, which should be the one closest to or even in a feasible region. As in Solver 1, variable values x_i in the sample points are constrained to be in the interval [0,1].

Recall that each fmincon iteration requires about n function evaluations. Thus, each restart with k random sample points results in a total of about $k + n$ function evaluations. In contrast, k restarts in Solver 1 result in a total of about kn function evaluations.

3.4 Solver 3: Local random restart of local searches

Similar to Solver 1, Solver 3 calls fmincon from a starting point randomly generated in a certain search region. The difference is that when fmincon fails to find a feasible solution, Solver 3 restarts fmincon from a random point close to the stopping point of the failed run. Specifically, in our experiments, the restart points are generated in the interval $[x_i^* - 0.25, x_i^* + 0.25]$, where x_i^* is the value of the variable in the stopping point of the failed run. Note that Solver 1 always restarts inside the hypercube $[0, 1]^n$, whereas Solver 3 may restart outside the hypercube.

The motivation for this solver is that the local search may have been on the right path, but got stuck in an adverse neighborhood of the function, e.g., a flat region. Instead of completely abandoning the prior search as in a global restart, a local restart tries to make use of history by searching for a better point from where to continue search with the local solver.

4 Experimental Results

We ran experiments with the three continuous formulations on randomly generated 3-SAT problems with n ranging from 10 to 60. Figure 4 shows the indicative results of the three formulations with 25 and 50 variables and the constraint ratio (ratio of constraints to variables) varying from 1 through 3.4. 100 random instances were solved. Each instance was solved once from a random starting point. The same random instances were solved by Solver 2 and 3, from the same starting points as well. The median numbers of iterations in solving the 100 instances are shown. Results of problems with different number of variables are similar.

We use the number of fmincon iterations as a measure of the computational cost for solving the problems, and we plot this cost in relation to the constraint ratio (ratio of constraints to variables). Each iteration typically involves on the order of n function calls. Thus, total computation time is proportional to n times the number of iterations. For Solver 2, computation time is also increased by a small value (roughly equivalent to one iteration if $k = n$) to account for the sample evaluations between restarts.

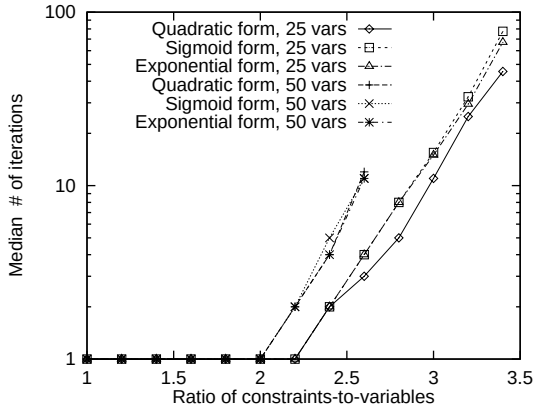

Figure 4: Complexity of continuous problems in the three formulations using Solver 1.

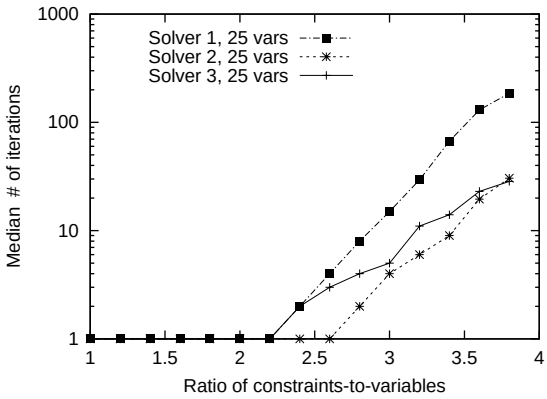

Figure 5: Complexity of continuous problems (25 variables) in the exponential formulation solved by the three solvers.

In Figure 4, the median number of iterations versus constraint ratio is plotted for n = 25 and 50 variables for the three different continuous formulations of the 3-SAT problem. There is virtually no difference between the different mappings, a result validated for other values of n. This result indicates that when predicting the difficulty in solving a continuous SAT problem, the results are independent of how the problem is mapped to the continuous domain. One reason may be that the three formulations are not very different within the hypercube $[0, 1]^n$, where the searches mostly take place.

A second observation is that the curves exhibit great similarity to each other and to discrete 3-SAT results. Slightly above a constraint ratio of 2 both the 25 and 50 variable problems become exponentially more difficult, while below this ratio the problems are relatively easy to solve. Thus, the ratio of constraints to the number of variables is an important parameterization of the problem.

This transition from polynomial to exponential cost is analogous to a transition in discrete 3-SAT [Hogg and Williams, 1994; Coarfa *et al.*, 2001]. In particular, an approximate theory predicts a transition from polynomial to exponential scaling of costs occurs just when there are enough clauses to make the expected number of consistent partial assignments no longer monotonic as a function of the number of assigned variables. For random k-SAT, this occurs when the clause to variable ratio μ satisfies $k\mu = (2^k - 1)\ln 2$ (Eq. 8 of [Williams and Hogg, 1994]). For 3-SAT, this gives $\mu = 1.62$.

These results are qualitatively quite similar to those found in discrete 3-SAT problems [Selman *et al.*, 1996]. The analyses of both the discrete and continuous problems indicate that the complexity is independent of the number of constraints when the constraint-to-variable ratio is less than 2. The discrete case undergoes a transition from weak dependence of complexity to strong dependence at a slightly higher ratio than the continuous case. In both continuous and discrete cases, the exponential rate of increase is approximately independent of the number of variables and depends only on the ratio of constraints to variables. The marked similarity

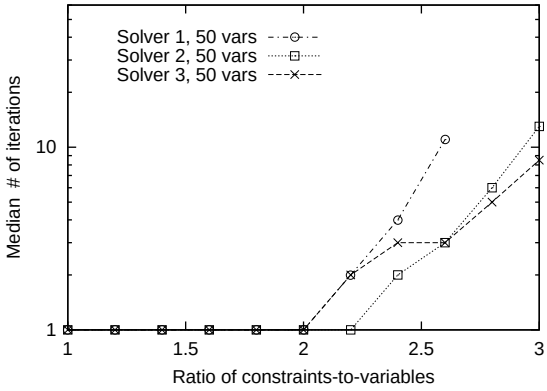

Figure 6: Complexity of continuous problems (50 variables) in the exponential formulation solved by the three solvers.

between the discrete and continuous cases is strong evidence that the observed complexity of solving the 3-SAT problems are characteristic of the problem and not the representation of the problem.

We examine the effect of the three different algorithms in Figure 5 for $n = 25$ and in Figure 6 for $n = 50$, with $k = n$ for Solver 2. All three algorithms exhibit a transition to the exponential behavior at about the same constraint ratio for the various numbers of variables.

The exponential slope of the median number of iterations does appear to have a dependence on the algorithm and therefore cannot yet be considered solely a characteristic of the problem. The median number of iterations increases more slowly for Solvers 2 and 3 than Solver 1. That the transition between easy and hard problems appears to be independent of the algorithm again suggests that the observed complexity behavior is characteristic of the problem. However, the different slopes represent significantly different solution times for problems above the critical constraint ratio. The solution times can vary as much as a factor of ten between the various algorithms.

5 Conclusion and Future Work

In this work, the complexities for solving continuous and discrete versions of SAT problems exhibit highly similar behavior as a function of problem characteristics such as the number of variables and the number of constraints. Importantly, this behavior is independent of the mapping between the discrete and continuous problems and largely independent of the algorithm used to solve the continuous problem.

One major difference between the discrete and continuous problems is that, for the discrete case, complete algorithms exist that can eliminate infeasible regions, and it is possible to prove that there are no solutions. This fact causes highly constrained discrete problems to become easier after reaching a maximum difficulty where neither search for possible solutions or elimination of infeasible regions works well. No such simplification currently exists for continuous problems, and problems currently continually become more difficult with increasing constraint ratio.

The marked similarity between the continuous and discrete problem complexity provides further strong evidence that the complexity transitions are characteristic of the problem rather than the algorithm or representation of the problem. Moreover, much of the work on discrete optimization probably applies in some form to the continuous SAT problems.

Our results confirm the conclusion that methods such as SQP are only appropriate for underconstrained problems. It is also clear that, when using these algorithms for problems arising in real-time applications, one would want to stay to the left of the exponential part of the complexity curve, since it becomes quickly impossible to guarantee sensible time bounds.

There are several variations on the algorithms presented in this paper that are promising candidates for further analysis. As a next step, we plan to evaluate Solver 1 with a larger window for the random restart points in order to increase the chance to be in or close to a feasible region. From early experiments, we have indications that a window such as [-1,2] improves the solver's performance. (The sigmoid formulation, for one, is largely flat outside that window, and thus a much larger window probably won't improve performance.) Another logical extension is to make the same change to Solver 3 as Solver 2 is to Solver 1, i.e., to sample multiple points before a restart. Another direction for research is the adaptive variation of problem and algorithm parameters over the run of a solver, such as increasing α and β for the sigmoid and exponential problem formulations, starting from low values, in order to speed up convergence, and the adaptive variation of k in Solver 2 to minimize the overhead.

Longer term, we plan to go beyond 3-SAT-like problems and to analyze and compare complexity results for random nonlinear constraint satisfaction and optimization problems.

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.

[Coarfa *et al.*, 2001] Cristian Coarfa *et al.* Random 3-SAT: The plot thickens. Technical report, Dept. of Computer Science, Rice Univ., 2001. Available at www.cs.rice.edu/~vardi/papers.

[Gu *et al.*, 1997] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the satisfiability (sat) problem: A survey. In Ding-Zhu Du, Jun Gu, and Panos Pardalos, editors, *Satisfiability Problem: Theory and Applications*, pages 19–152. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1997.

[Hogg and Williams, 1994] Tad Hogg and Colin P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377, 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.

[Hooker, 1988] J. N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

[Kamath *et al.*, 1990] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resend e. Computational experience with an interior point algorithm on the satisfiabil ity problem. *Annals of Operations Research*, 25:43–58, 1990.

[Powell, 1983] M. J. D. Powell. Variable metric methods for constrained optimization. In A. Bachem, M. Grotschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 288–311. Springer-Verlag, 1983.

[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.

[Strogatz, 1994] S. H. Strogatz. *Nonlinear Dynamics and Chaos: with Applications in Physics, Biology, Chemistry and Engineering*. Addison Wesley, 1994.

[Williams and Hogg, 1994] Colin P. Williams and Tad Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[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.