Modular Robot Control and Continuous Constraint Satisfaction

Markus P.J. Fromherz, Tad Hogg, Yi Shang
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{fromherz,hogg,yshang}@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 constraint satisfaction is at the core of many real-world applications. One example is in the control of modular, hyper-redundant robots, which are robots with many more degrees of freedom than required for typical tasks. Casting the control problem as a constraint problem is a promising approach for robustly handling a variety of non-standard constraints found in such robots. However, before we can scale to the many degrees of freedom and nonlinearities of this system and deploy constraint solvers for embedded, real-time control, we need to better understand the complexity issues arising in these problems. In this paper, we first present a parametric model for robotic control. We then study the complexity of related but simpler problems by analyzing two classes of artificial constraint satisfaction problems inspired by (discrete) 3-SAT problems, which have a strong relation between structure and search cost. With this, we also propose a generic benchmarking model for continuous constraint satisfaction problems.

1 Introduction

Many important real-world problems in domains such as planning, control, reconfiguration, and fault diagnosis can be regarded as constrained optimization problems [Bondarenko *et al.*, 1998]. One example is modular robot control. Modular robots consist of many interchangeable robotic modules connected into a mechanical and functional assembly. Interest in modular robots has increased recently due to their potential for robustness, reduced costs, and wide range of applicability particularly in highly constrained environments [Yim, 1994; Rus and Vona, 1999; Kotay *et al.*, 1998]. Typically, there are only a small number of module types, each module with limited motion capability. The flexibility of modular robots is achieved through the large number of modules, expected to range from dozens to thousands, and their many possible configurations. Because of the large number of degrees of freedom of the redundant modular limbs, such modular robots are also called hyper-redundant. In this paper, we focus on the control of a subclass of such robots, namely hyper-redundant manipulators that consist of a chain of modules with rotational joints [Yim, 1994] (Figure 1).

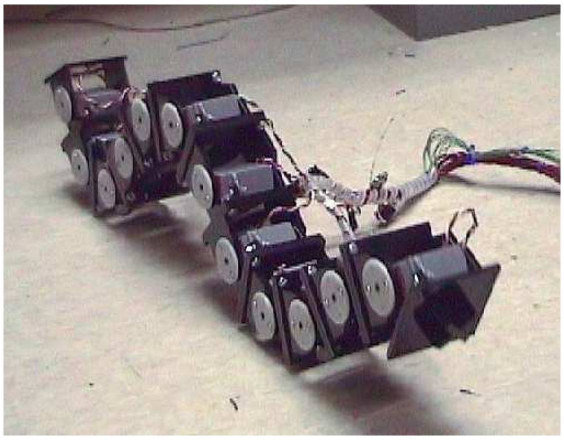

Figure 1: Modular robot prototype PolyBot (12 joints).

Realizing the potential of modular robots requires robust, versatile and scalable control algorithms satisfying many changing joint and force constraints. A control approach should not only handle the many degrees of freedom with limited computational resources in real time, but should also attempt to utilize the redundancy in an optimal way. For example, the robot may not only be asked to achieve typical goals such as reaching a goal position and orientation with its end effector, but also to minimize velocities, distribute torques equally, and avoid complex obstacles. At the same time, each module has to obey its physical limits, such as limits on joint angles and motor torques. Finally, to simplify programming different configurations of modular robots, the software should also be generic with respect to constraints and goals.

To robustly handle a variety of constraints for the many degrees of freedom found in modular robots, we develop a parametric model and cast the control problem as a constrained optimization problem. The constraints represent the (mainly physical) limits of the robot, while the objective function expresses the various goals of interest. Furthermore, we assume that end-effector goals (e.g., position and orientation) are given by a higher-level controller such as a path planner, and we focus on the problem of *actuation allocation*, the problem to finding optimal solutions for the joint angles at each time step.

Optimization problems in real applications such as modular robot control are often best solved in the continuous domain in order to avoid excessive nonlinearities, either because the underlying physical system is continuous or because discrete implementations exhibit abrupt changes in outputs for small changes in inputs [Strogatz, 1994]. However, effective constraint-based techniques must handle the complexity of real-world continuous constraint problems. In particular, we believe that solvers must be able to dynamically *adapt* to the structure of the problem. Our thesis is that knowing about general problem properties characterized by, for example, complexity phase transitions [Hogg *et al.*, 1996b] should help constraint solvers in this task. Toward this end, this paper extends some of the progress made in understanding complexity results for discrete constrained satisfaction problems, particularly (discrete) satisfiability (SAT) problems, to continuous problems.

The rest of this paper is structured as follows. In the next section, we review related work on the control of redundant manipulators and present our parametric robot control model as an example for modeling a real-world application as a constraint problem. In Section 3, we present several related but simpler formulations of continuous constraint problems, with the goal of understanding the relevant characteristics for predicting problem complexity. In Section 4, these problems are solved and analyzed using two variants 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. While continuous SAT-like problems exhibited complexity similar to the behavior exhibited by discrete problems, more general continuous problem formulations seem to require the use of additional parameters to understand solving behavior. Section 5 closes with conclusions and future work.

2 Hyper-redundant Manipulator Control

Previous work on the control of redundant manipulators has focused on their kinematics, involving the computation and inversion of the Jacobian mapping from joint space to the end-effector space [Khatib, 1987; Ghosal and Roth, 1988; Chiu, 1988]. These approaches typically include few or no constraints and have been applied to robots with a small number of degrees of freedom (less than 10). Attempts have been made to include more constraints by extending Jacobian methods with projected gradients of the constraints [Baillieul, 1986] or numerically computed gradients [Klein *et al.*, 1995]. These methods can work reasonably well for systems with a small number of joints and a few well-characterized constraints. However, modular robot control problems tend to have too many degrees of freedom and mixed task space, joint space, and dynamic inequality constraints to solve such systems in real time. Also, because these methods tend to require a detailed constraint analysis, it is difficult to robustly adjust the priorities of different constraints over time as robot environment and pose change. Moreover, since the solutions are computed in velocity space, position errors can build up over time.

In order to deal with constraints, certain kinds of dynamic constraints can be included in an unconstrained form using higher-order differential forms [Agrawal and Faiz, 1998], which can then be used to elegantly provide optimal motion planning and control [Gokce and Agrawal, 1999]. However, mixed task space and joint space inequality constraints do not map easily into these higher-order forms.

Control of hyper-redundant robots using continuous backbone curves has been well studied [Chirikjian and Burdick, 1995]. This approach scales very well to large numbers of modules and is of particular interest in hierarchical control approaches. Again, this approach has only been studied with kinematic constraints.

Taking into account constraints such as torque limits is particularly relevant for modular robots. Traditional robots are designed such that their joint torque limits are never exceeded (e.g., by making the base joints strong and the end joints lightweight). In a modular robot, all modules are similar and often relatively weak.

In contrast to traditional approaches to manipulator control, in our model, we separate constraints, goals, and control algorithms from each other and include a larger class of constraints.

2.1 Manipulator Model

For the purpose of this paper, we focus on a robotic manipulator that consists of a chain of n links. This manipulator is attached to a base at one end (link 1), and the primary task is to reach a goal position, or follow a goal trajectory, with the other end (link n). Such a manipulator may, for example, be an arm (which is attached to a table or robot body and whose "hand" end grasps or pushes other objects) or a leg (which is attached to a robot body and whose "foot" end is moving over the ground during locomotion). Note that for now we consider only statically stable configurations, and, due to the relatively slow movement of the joints, we are not concerned with dynamical constraints.

Each manipulator link i has its own local coordinate system, or *frame*, with unit vectors $\hat{X}_i$, $\hat{Y}_i$, and $\hat{Z}_i$ defining the three axes. In addition, we have a base frame or frame 0, e.g., a table or body, to which the first link is attached, as well as the global frame, the world coordinate system. Following robotics conventions [Craig, 1989], the kinematics of each link is defined by its Denavit-Hartenberg parameters $\langle \alpha_i, a_i, \theta_i, d_i \rangle$, where (cf. Figure 2)

- twist α_i is the angle between $\hat{Z}_{i-1}$ and $\hat{Z}_i$ measured about $\hat{X}_{i-1}$,
- length a_i is the distance from $\hat{Z}_i$ to $\hat{Z}_{i+1}$ measured along $\hat{X}_i$,
- rotation θ_i is the angle between $\hat{X}_{i-1}$ and $\hat{X}_i$ measured about $\hat{Z}_i$, and
- extension d_i is the distance from $\hat{X}_{i-1}$ to $\hat{X}_i$ measured along $\hat{Z}_i$.

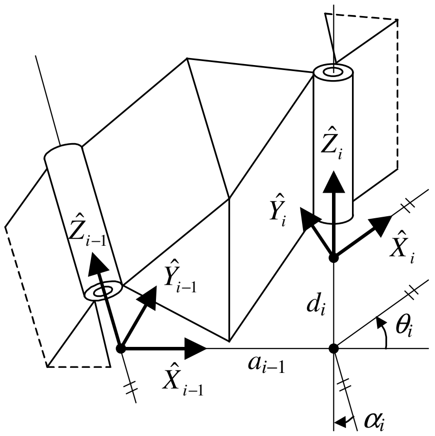

Figure 2: Parameters for coordinate transformation from frame $i-1$ to frame i.

θ_i is a variable for revolute link joints, and d_i is a variable for prismatic link joints. In this paper, we restrict ourselves to revolute joints, and d_i is always 0. Also, the origin o_i of frame i is located in the joint of link i.

These parameters define a link's frame with respect to its predecessor's frame in the chain of links. More precisely, these parameters define a *homogeneous transform matrix* ${}^{i-1}_{i}T$, a rotation plus a translation, that maps a point ${}^{i}p$ defined in frame i to point ${}^{i-1}p$ defined in frame $i-1$ by the multiplication ${}^{i-1}p = {}^{i-1}_{i}T\ {}^{i}p$ [Craig, 1989]. (Here, a point $(x\ y\ z)$ is represented by its homogeneous coordinates $p = [x\ y\ z\ 1]^{\mathrm{T}}$, with the superscript "T" indicating the transpose. A point p or vector $\mathbf{r}$ is defined in the global frame, unless a prefix superscript identifying the local frame is given, as in ${}^{i}p$ and ${}^{i}\mathbf{r}$.) The transform matrix is defined as

$$\begin{aligned} {}^{i-1}_{i}T &= \begin{bmatrix} {}^{i-1}_{i}R & -{}^{i-1}_{i}R\ {}^{i-1}o_i \\ 0\,0\,0 & 1 \end{bmatrix} \qquad (1) \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_i & \cos\theta_i\cos\alpha_i & -\sin\alpha_i & -\sin\alpha_i d_i \\ \sin\theta_i\sin\alpha_i & \cos\theta_i\sin\alpha_i & \cos\alpha_i & \cos\alpha_i d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

where ${}^{i-1}_{i}R$ is the rotation matrix from frame $i-1$ to i, and ${}^{i-1}o_i$ is the origin of frame i given in (the coordinate system of) frame $i-1$. The three columns of the rotation matrix define the *unit vectors* $\hat{X}_i$, $\hat{Y}_i$ and $\hat{Z}_i$ of frame i in frame $i-1$. We will also use the functional denotation ${}^{i-1}_{i}T = T(\alpha_i, a_{i-1}, \theta_i, d_i)$ to describe the matrix of Eq. (1).

Transform matrices can be composed by multiplying them. In particular, the transformation from base frame 0 to frame i is given by

$${}^{0}_{i}T = {}^{0}_{1}T\ {}^{1}_{2}T \ldots {}^{i-1}_{i}T = {}^{0}_{i-1}T\ {}^{i-1}_{i}T$$

For example, a point ${}^{i}p$ defined in frame i, i.e., in the local coordinate system of link i, has position ${}^{0}p = {}^{0}_{i}T\ {}^{i}p$ in the base coordinate system. As another example, the origin of frame i is given in the base frame by ${}^{0}o_i = {}^{0}_{i}T[0\ 0\ 0\ 1]^{\mathrm{T}}$. In this paper, the base frame is always identical to the global frame, i.e., ${}^{0}p = p$.

In addition to the kinematics parameters, each link has the following parameters:

- link mass m_i,
- center of mass position ${}^{i}c_i$ (in the link's frame),
- joint angle limits $\theta_{\min,i}$ and $\theta_{\max,i}$,
- maximum rotational velocity $\omega_{\max,i}$ achievable by the joint, and
- maximum motor torque $\tau_{\max,i}$.

2.2 Manipulator Constraints

The manipulator's only actuator variables are the joint angles $\theta_1, \ldots, \theta_n$, and thus all constraints ultimately have to be expressed as constraints on these angles. Given the links' parameters, we take into account up to four types of constraints: joint angle limits, torque limits, velocity limits, and obstacle avoidance.

Angle limits. The first type of constraint states that a joint's movement is restricted by the mechanics of the link in either direction of the rotation and is defined trivially as

$$\theta_{\min,i} \leq \theta_i \leq \theta_{\max,i} \ \text{ for all } i = 1, \ldots, n$$

Torque limits. For each joint i, the links from i through n together exert a torque onto joint i, which is limited by the strength of the joint. As noted before, this problem has traditionally not been addressed in manipulator control, and related work has been restricted to 2D manipulators [Agrawal and Garimella, 1994; Gokce and Agrawal, 1999].

A torque is determined primarily by the gravitational force (applied at the center of gravity of links i through n) and the moment arm (the vector from the joint axis to where the force is applied). As a vector, this torque is both perpendicular to the moment arm and tangential to the rotation (Figure 3). The torque τ_i on joint i can be computed as

$$\tau_i = \hat{Z}_i \cdot (\mathbf{R}_i \times \mathbf{F}_i)$$

where $\hat{Z}_i$ is the axis of rotation (cf. Figure 2), $\mathbf{R}_i$ is the vector from joint i to the center of mass C_i of links i through n, and $\mathbf{F}_i$ is the gravitational force on links i through n. The magnitude of $\mathbf{F}_i$ is $F_i = \sum_{j=i}^{m} m_j g$ (where g is the gravitational constant), and $\mathbf{F}_i$ is directed vertically down, i.e., $\mathbf{F}_i = [0\ 0\ -F_i]^{\mathrm{T}}$.

The center of mass C_i of links i through n is the weighted average of the centers of mass of each link relative to the origin of link i, and thus $\mathbf{R}_i$ can be computed as follows.

$$\mathbf{R}_i = \frac{\sum_{j=i}^{n} m_j (c_j - o_i)}{\sum_{j=i}^{n} m_j}$$

Recall that $c_i = {}^{0}_{i}T\ {}^{i}c_i$ is the center of mass of link i and o_i is the link's origin (both in the base frame). Note that τ_i and $\mathbf{R}_i$ depend on all the joint angles, which we indicate by writing $\tau_i(\theta)$. The nonlinear torque constraint is then defined as

$$\tau_i(\theta) \leq \tau_{\max,i} \ \text{ for all } i = 1, \ldots, n$$

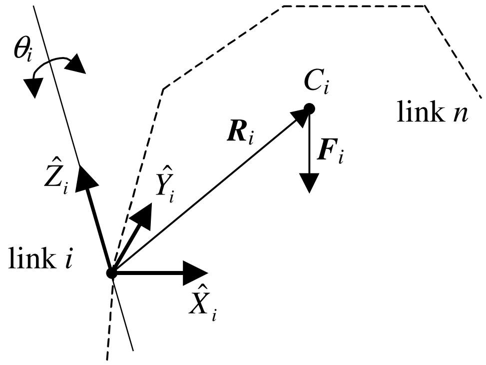

Figure 3: Center of gravity C_i, gravity force $\mathbf{F}_i$, and moment arm $\mathbf{R}_i$ for links i through n.

Velocity limits. For the movement from a previous position, we define as a constraint the maximum velocity achievable by each joint, i.e., $|\omega_i| \leq \omega_{\max,i}$ for all i. We use a linear approximation for joint velocity, i.e., $\omega_i = (\theta'_i - \theta_i)/dt$, where dt is the time step for control and θ'_i is the previous angle value for joint i. Thus, this constraint is defined as

$$\theta'_i - \omega_{\max,i}dt \leq \theta_i \leq \theta'_i + \omega_{\max,i}dt \quad \text{for all } i = 1,\ldots,n$$

Obstacle avoidance. As a final example, we require the robot arm to avoid known obstacles. The comprehensive form of this constraint, including avoiding collision of the arm with itself, is beyond the scope of this paper. Here, consider the constraint for specific links i with origins ${}^io_i = [x_i\ y_i\ z_i\ 1]^{\mathrm{T}}$ to avoid spheric obstacles at positions $p_j = [x_j\ y_j\ z_j\ 1]^{\mathrm{T}}$ with radius r_j, which can be defined as

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \geq a_i + r_j$$

where a_i is an approximation of the link's size. Note that the position $o_i = {}^0_iT[0\ 0\ 0\ 1]^{\mathrm{T}}$ depends on the joint angles $\theta_1,\ldots,\theta_i$.

As in many real-world applications, the number of constraints is proportional to the number of variables for most of the constraints. Obstacle avoidance as formulated above can be used to set the numbers of variables and constraints independently and thus is of particular interest for testing the effect of different ratios on problem complexity. Figure 4 shows a sample solution to a problem with 20 links, nine randomly placed obstacles, and an additional constraint that requires the end-effector to reach point $(0\ 0\ 17)$.

2.3 Task Model

In a typical task, the main goal of the robot manipulator is to reach or follow a target position with its end link. There may be secondary goals, however, such as also achieving a target orientation with the end link, minimizing the difference from a previous position, and distributing actuation equally. At the same time, the manipulator has to satisfy its constraints, and

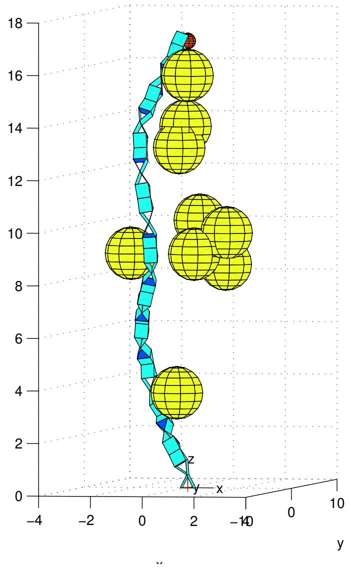

Figure 4: Hyper-redundant manipulator with 20 links and nine obstacles reaching for point $(0\ 0\ 17)$.

it may not always be able to achieve its goals exactly. Thus, we define the actuation allocation task as the constrained optimization problem

$$\begin{array}{rl} \text{minimize} & h(\theta) \\ \text{subject to} & c(\theta) \end{array}$$

where $\theta = \{\theta_1,\ldots,\theta_n\}$ are the free variables. The constraints $c(\theta)$ are those presented above. The objective function is presented in the balance of this section. In a control application, this problem is solved for the joint angles once per control cycle, which are then sent to the modules for low-level control.

The total objective function is a weighted, normalized sum of the individual goal functions $h_i(\theta)$:

$$h(\theta) = \frac{\sum_i w_i \frac{h_i(\theta)}{q_i}}{\sum_i w_i}$$

Thus, both the individual goal functions and the entire sum are normalized to be in the range [0,1]. The weights w_i determine the relative priority of the goals, and the scaling factors $q_i = \max_\theta(h_i(\theta))$ normalize the goal functions (not shown in the following).

In the following, we present two individual goals.

Position goal. Given a goal position $p_{\mathrm{g}} = [x_{\mathrm{g}}\ y_{\mathrm{g}}\ z_{\mathrm{g}}\ 1]^{\mathrm{T}}$ for the end effector (the end of link n), the goal function is

$$h_1(\theta) = \sqrt{(x_{\mathrm{g}} - x_{\mathrm{e}})^2 + (y_{\mathrm{g}} - y_{\mathrm{e}})^2 + (z_{\mathrm{g}} - z_{\mathrm{e}})^2}$$

where $p_{\mathrm{e}} = [x_{\mathrm{e}}\ y_{\mathrm{e}}\ z_{\mathrm{e}}\ 1]^{\mathrm{T}} = {}^0_nT\ {}^n_{\mathrm{e}}T[0\ 0\ 0\ 1]^{\mathrm{T}}$ is the position of the end effector, with ${}^n_{\mathrm{e}}T = T(0, a_n, 0, 0)$ the transform matrix from the origin of link n to its end (a translation by a_n along $\hat{X}_n$; cf. Eq. (1)).

Orientation goal. We currently allow one to define a goal twist α_g and rotation θ_g for the end effector. These angles define a frame ${}^0_gT = T(\alpha_g, 0, \theta_g, 0)$, which can be related to the frame ${}^0_eT$ of the end-effector with a single axis and a single rotation angle ${}^e_g\Theta$ using Euler's theorem on rotation [Craig, 1989]. This angle is computed from the rotation matrix as

$$ {}^e_g\Theta = arccos\left(\frac{\mathrm{Tr}({}^e_gR) - 1}{2}\right) \qquad (2)$$

where $\mathrm{Tr}({}^e_gR) = \sum_{i=1}^{3} r_{ii}$ is the trace of ${}^e_gR$ (r_{ij} being the elements of ${}^e_gR$). We define the goal function for an orientation goal by this rotation angle:

$$h_2(\theta) = {}^e_g\Theta$$

See an earlier paper for details on these and other objectives, such as minimal actuation energy and equal actuation distribution [Fromherz *et al.*, 1999].

3 Continuous CSPs

The modular robot control problem offers a variety of complex constraints with various parameters that might influence the complexity of the constraint problem. In order to come to an understanding of the relevant characteristics for analyzing and predicting the complexity of such continuous nonlinear constraint problems, we have started to model and study a series of simpler problems, which we present in this section.

3.1 SAT-like Continuous CSPs

In the past ten years, significant progress has been made in understanding problem complexity of the (discrete) satisfiability (SAT) problem [Hogg *et al.*, 1996a]. Thus, in a first step, we have begun to analyze the complexity of continuous, SAT-like constraint satisfaction problems and compared them to results from discrete SAT problems [Shang *et al.*, 2001].

SAT problems can be formulated as discrete or continuous, constrained or unconstrained, optimization problems. In the constraint programming approach, SAT problems have been solved as integer programming problems using various constraint programming algorithms [Gu *et al.*, 1997], and as nonlinear continuous constrained optimization problems using Lagrange-multiplier methods [Chang and Wah, 1995]. In contrast to previous work that aims at solving SAT problems faster, our focus is on gaining a better understanding of the computational cost of continuous constraint problems. We use 3-SAT-like problems in our study because of the rich empirical and theoretical results on discrete SAT problems.

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 incomplete algorithms such as GSAT [Selman *et al.*, 1992], the complexity curve still exhibits the behavior for soluble problems with varying degrees of constraint (such algorithms are not useful for problems with no solutions).

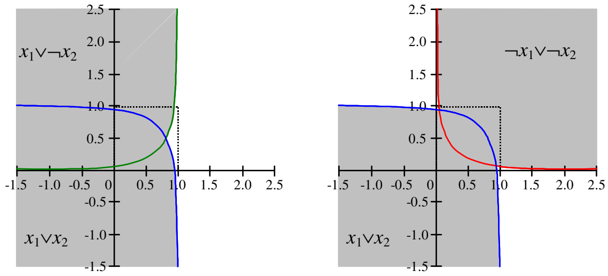

Figure 5: 2-D feasible spaces of the exponential formulation for two 2-SAT problems.

We have compared these results with a study of several continuous, 3-SAT-like problem formulations and (incomplete) randomized continuous constraint solving algorithms [Shang *et al.*, 2001]. In these formulations, each positive literal x in a 3-SAT clause is converted to a continuous function $f(x)$, each negative literal $\neg x$ is converted to $f(1-x)$, disjunction (e.g., $x_1 \vee x_2$) is represented as addition in the constraint (e.g., $f(x_1) + f(x_2)$), and 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 function f and constant c depend on the formulation. For example, in an exponential formulation, $f(x) = 2^{\beta(x-1)}$ and $c = 1$ (cf. Figure 5). (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 at 0.5.) Thus, a clause $x_1 \vee x_2 \vee \neg x_3$ is transformed into the nonlinear constraint

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

We have studied three different problem formulations with several variants of a randomized sequential quadratic programming (SQP) solver [Shang *et al.*, 2001]. Figure 6 shows representative 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, using the first of the solvers. The median numbers of iterations of 100 random runs are shown. This figure shows that the curves exhibit 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 seems to be an important parameterization of the problem.

These results are qualitatively quite similar to those found in discrete 3-SAT problems [Selman *et al.*, 1996]. 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 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 rate of exponential increase is similar in both cases. This similarity 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.

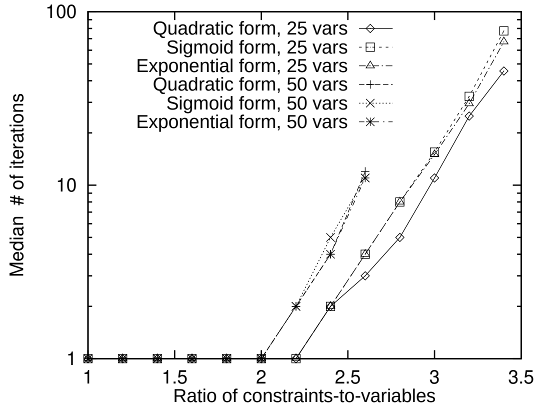

Figure 6: Complexity of continuous 3-SAT problems in three formulations using Solver 1 (cf. Section 4).

3.2 Sine-based Continuous CSPs

The SAT-like continuous constraint problem formulations allow us to connect our work to the studies of (discrete) SAT problems, but they are far removed from real applications such as the robot control problem. In particular, the robot control problem is representative of problems with spatially defined constraints using trigonometric functions that are often combined in products (not just sums as in our SAT formulations). In this section, we therefore propose a test-case generator for sine-based continuous CSPs.

We feel there is an unfulfilled need for generic benchmark problem sets in the domain of continuous constraint problems. In various research areas, many nonlinear constrained optimization/satisfaction methods have been developed in the past, including mathematical programming methods, evolutionary methods, and simulated annealing methods. Performance of these methods often varies across different application domains and even from one problem instance to another. Although efforts have been taken to evaluate these methods, they are mostly done within respective research communities. For example, in the mathematical programming community, several test problems sets have been proposed, such as CUTE (Constrained and Unconstrained Testing Environment) [CUTE, 2001] and COPS (Constrained Optimization ProblemS) [Bondarenko *et al.*, 1998], which are collections of nonlinear constraint problems from real applications. Unfortunately, while the number of variables is often a parameter in these problems, the ratio of variables and constraints cannot be changed in either of these problem sets. Since previous phase transition work on random SAT problems concluded that the clause-to-variable ratio is an important parameter in determining the difficulty of SAT problems, changing that ratio is an important capability for our purpose.

In the community of evolutionary computing, various evolutionary algorithms have been proposed for complex constraint problems, including non-differentiable or discontinuous problems, and have been evaluated on various test cases such as problems G1-G11 proposed by Michalewicz and Schoenauer [Michalewicz and Schoenauer, 1996]. Although these test cases include objective functions of various types with various numbers of variables and different numbers and types of constraints, they failed to provide meaningful conclusion on what characteristics make a constraint problem difficult for an evolutionary technique, or in general for any solving technique, and the authors called for a new suite of test problems. These flexible test problems should allow one to bring different optimization techniques developed in various scientific communities together and compare them on a level ground. The test problems should be useful in investigating the complexity of constraint problems as well as the computational complexity of various solving techniques.

Our proposed generator is capable of generating random test problems with different characteristics, in a form similar to randomly generate k-SAT problems. A k-SAT problem is characterized by three parameters: the number of variables, the number of clauses, and the number k of literals in each clause. The parameters of our generator $G(n, m, k, \theta)$ are

- n, number of variables,
- m, number of constraints,
- k, number of variables in each constraint, and
- $\theta \in [-1, 1]$, a threshold to determine the tightness of each constraint.

We define the tightness of a (continuous) constraint as the ratio of feasible to total problem space. All the parameters of the generator can be changed independently. In our current implementation, all constraints are generated based on the same base function with randomly generated coefficients. One base form reminiscent of the SAT-like formulations is

$$\frac{1}{k}\sum_{i=1}^{k} a_i \sin(2\pi\alpha b_i x_{j_i} + 2\pi c_i) \leq \frac{1}{k}\theta \sum_{i=1}^{k} a_i \qquad (3)$$

where $\{j_1, \ldots, j_k\} \subseteq \{1, \ldots, n\}$ 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 specifies the frequency range. A larger value of α makes the feasible region specified by the constraint more discontinuous. $\alpha = 1$ is used in our experiments. θ specifies the threshold for satisfying the constraint. The smaller its value, the easier the constraint is to satisfy. For $\theta = 0$, on average half of the search space is feasible for the constraint. Both the left and right hand side of Eq. (3) are normalized to have values between -1 and 1.

Figure 7 shows the 3-D plot of four random constraints generated according to Eq. (3) with $k = 2$ and $\theta = 0.35$. Figure 8 shows the 2-D feasible spaces formed by the combination of the same four random constraints with $\theta = 0.35$ (left diagram). For comparison purposes, Figure 8 also shows the 2-D feasible spaces formed by the combination of the same four constraints except for $\theta = 0$ (right diagram). When $\theta = 0$, each constraint produces feasible regions smaller than those when $\theta = 0.35$. Therefore, the feasible space (corresponding to regions inside the contour lines in Figure 8) formed by the combination of the four constraints is smaller.

In discrete 3-SAT problems, each clause makes $\frac{1}{8}$ of the search space infeasible. In other words, the tightness (ratio of feasible space) is exactly $\frac{7}{8}$ for each 3-SAT constraint. By

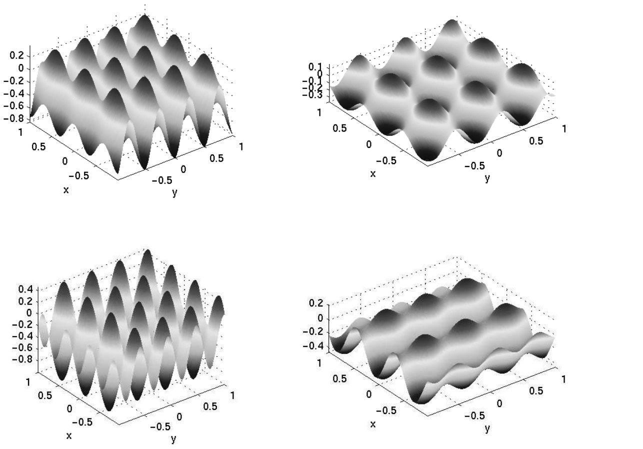

Figure 7: 3-D plot of four random constraints in the form of Eq. (3) with $\theta = 0.35$.

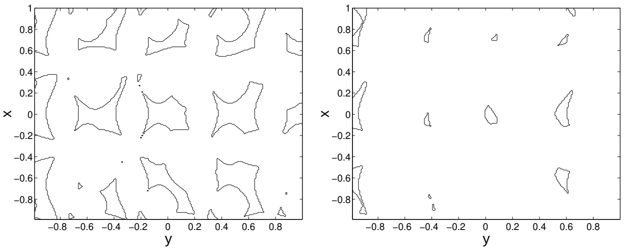

Figure 8: 2-D feasible spaces (regions inside the contour lines) formed by the combination of four random constraints with $\theta = 0.35$ (left) and $\theta = 0$ (right).

setting $k = 3$, the generator $G(n, m, 3, \theta)$ generates problems with each constraint containing 3 randomly selected variables. By changing θ, we can control the difficulty of each constraint, which in turn affects the complexity of the whole problem. Although it is hard to find the exact θ value for each random constraint so that the constraint makes a fixed fraction of the search space infeasible, we can find an approximate θ value statistically. Figure 9 shows the average and standard deviation of the tightness of randomly generated constraints with different θ values. The data in the figure is obtained based on 1000 randomly generated constraints in the form of Eq. (3), and each constraint is sampled randomly at 1000 points in the region [-1,1]. It shows that $\theta = 0.57$ approximately gives a tightness of $\frac{7}{8}$. Thus, $\theta = 0.57$ is used in our experiments to compare with 3-SAT based continuous CSPs. For comparison purposes, we also try $\theta = 0.35$ in the experiments in which case the tightness of each constraint is approximately $\frac{3}{4}$, corresponding to 2-SAT like problems. When $\theta = 0.35$, each constraint is, on average, almost twice as hard to satisfy by random sampling as when $\theta = 0.57$.

The generator $G(n, m, k, \theta)$ can use other base functions besides Eq. (3) to generate constraints with different characteristics. In particular, the formulation in Eq. (3) allows a linear decomposition of the function for the variables involved. Most constraints of the modular robot control problem in Section 2 involve the *product* of the constraints' components (sines and cosines). For example, given any link k, the transform matrices for variables $\theta_1, \ldots, \theta_k$ and for variables $\theta_k, \ldots, \theta_n$ are multiplied for kinematic and torque constraints, respectively, on this link.

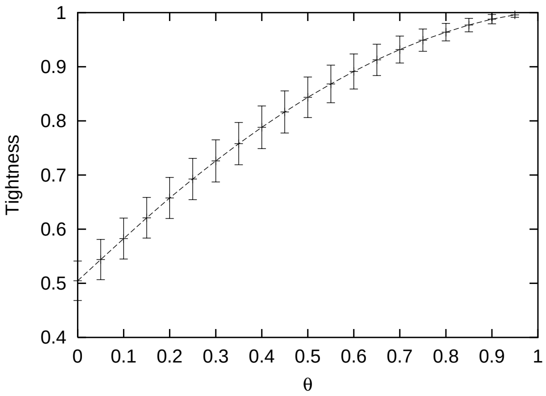

Figure 9: Average and standard deviation of the tightness of randomly generated constraints (Eq. (3)) with varying θ.

A possible base function to simulate such constraints is the following variation of Eq. (3):

$$\frac{1}{k}\prod_{i=1}^{k} a_i \sin(2\pi\alpha b_i x_{j_i} + 2\pi c_i) \leq \frac{1}{k}\theta\prod_{i=1}^{k} a_i \qquad (4)$$

with $\{j_1, \ldots, j_k\}$, a_i, b_i, and c_i chosen as in Eq. (3). This generic formulation is closer to the modular robot control problem. Of course, the summation and multiplication of variable components can also be mixed. We plan to investigate these differences and their effects on problem complexity in future work.

4 Experimental Results

In this section, we present our experimental results with a focus on the sine-based continuous CSPs. The programs are implemented in Matlab 6.0 and use sequential quadratic programming as the local search method.

4.1 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]. In each iteration an approximation is made of the Hessian of the Lagrangian function using a quasi-Newton updating method. This is then used to generate a quadratic programming subproblem whose solution is used to form a search direction for a line search procedure.

The function fmincon is a MATLAB implementation of SQP that finds the constrained minimum of a nonlinear multivariable function starting from an initial estimate. 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 functions 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 near 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.

4.2 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 the search space. If it stops without finding a consistent solution, fmincon is restarted from a new random point. This is repeated until a solution is found. For the continuous sine-based problems, variable values in the initial and restart points are constrained to be in the interval [-1,1].

We use the number of local solver iterations as a measure of the computational cost for computing the solution, and we plot this cost in relation to the constraint ratio (ratio of constraints to variables). Since each iteration involves on the order of n function calls, total computation time is proportional to n times the number of iterations.

For the sine-based continuous CSPs, we ran experiments on problems that were generated randomly by $G(n, m, k, \theta)$ ($k = 3$) with constraints in the form of Eq. (3). Figures 11 and 12 show the results for 25 and 50-variable CSPs and $\theta = 0$, 0.35, and 0.57. The median numbers of iterations of 100 random runs are shown. A phase transition from flat to exponential is observed when θ is large. The transition point is around 2 for the 25-variable problems with $\theta = 0.57$, and varies for other cases. The transition point is smaller for 50-variable problems and when θ is smaller.

In our experiments with Solver 1 solving SAT-like CSPs, fmincon often ran only a single iteration between restarts. This indicated that solver iterations were often wasted until a restart point was sufficiently close to a feasible region. For sine-based CSPs, Figure 10 shows that each restart can result in multiple iterations for small constraint ratios, but that the average number of iterations per restart reduces to 1 as the constraint ratio increases. (This trend is evident in the converging iteration and restart curves for the same problems.)

Our hypotheses is that, with regard to hard SAT-like continuous CSPs, the feasible region is either large and easy to find by fmincon (underconstrained cases), or small and hard to find, i.e., most of the search space is either flat or rugged, and gradients are not pointing to feasible regions in either

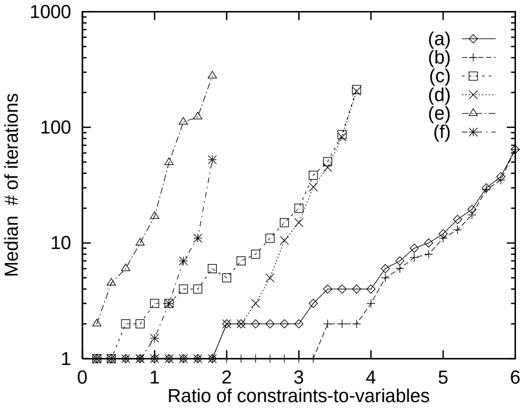

Figure 10: Average number of iterations versus average number of restarts of local search in solving sine-based continuous CSPs with 25 variables by Solver 1. (a) $\theta = 0.57$, (b) $\theta = 0.57$, no. of restarts, (c) $\theta = 0.35$, (d) $\theta = 0.35$, no. of restarts, (e) $\theta = 0$, and (f) $\theta = 0$, no. of restarts.

case. fmincon often cannot improve the Lagrangian function and so gives up. For the sine-based CSPs, the search space is smoother and functions change more gradually. Thus fmincon can continue for more iterations with improving Lagrangian function values. As the constraint ratio increases, feasible regions become smaller and most restart points are far from these regions in the search space. In addition, the Lagrangian function becomes increasingly rugged and is hard to improve through fmincon's quadratic approximation strategy. Hence fmincon immediately gives up after one iteration. When this happens, most runs of fmincon are wasted.

4.3 Solver 2: A Simple Heuristic in Generating Better Starting Points

The goal of our second solver is 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 the same range as the initial points, e.g., [-1, 1] for the sine-based problems.

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.

We show the results of the two search algorithms for sine-based CSPs in Figure 11 for 25-variable problems and Figure 12 for 50-variable problems. The median number of iterations versus constraint ratio is plotted for three different θ values, 0, 0.35, and 0.57 (cf. Eq. (3)). Both algorithms exhibit a transition to the exponential behavior at different constraint

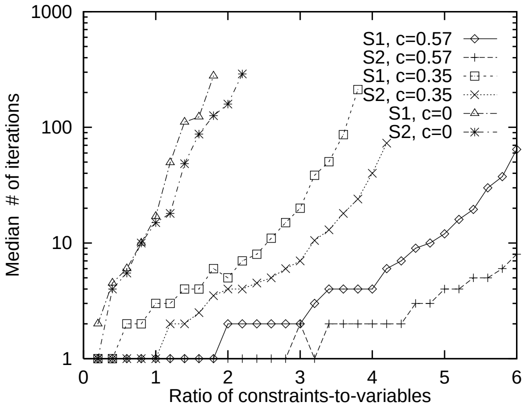

Figure 11: Complexity of continuous sine-based CSP (25 variables) solved by the two solvers (S1 for Solver 1 and S2 for Solver 2). The c in the diagram is the θ in Eq. (3).

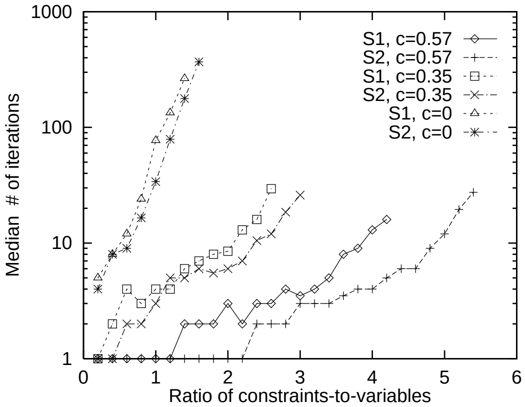

Figure 12: Complexity of continuous sine-based CSP (50 variables) solved by the two solvers (S1 for Solver 1 and S2 for Solver 2). The c in the diagram is the θ in Eq. (3).

ratios, although at slower slopes than what we observed for the SAT-like problems [Shang *et al.*, 2001]. Solver 2 always outperforms Solver 1 except at the occasional low constraint ratio (e.g., at 1.2 for the 50-variable problems and $\theta = 0.35$, Figure 12). This confirms results from the SAT-like continuous CSPs, were solution times above the critical constraint ratio could vary as much as a factor of ten between the two solvers.

Interestingly, the CSPs with $\theta = 0.35$ seem slightly more than twice as hard as the ones with $\theta = 0.57$, which can be explained by the fact that random constraints with $\theta = 0.35$ on average are tighter than any two random constraints with $\theta = 0.57$. The same observation can also be made for the CSPs with $\theta = 0$ compared to the ones with $\theta = 0.35$.

As Figures 11 and 12 illustrate, the constraint ratio is insufficient for predicting problem complexity. In particular, both the number of variables and the tightness of the constraints influence solver behavior. Methods to characterize constraint tightness for real problems such as the robot control problem seem to be an important research direction.

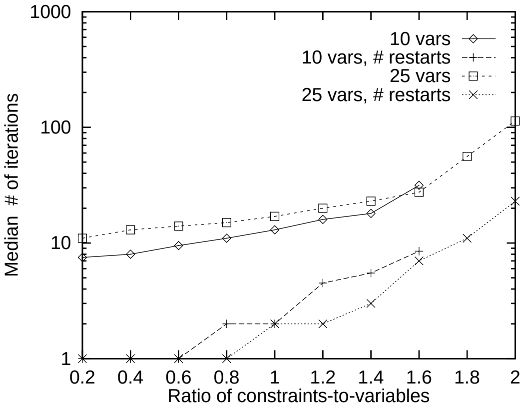

Figure 13: Complexity of the modular robot control problem solved by Solver 1.

5 Conclusion

In this paper, we present some empirical work on continuous CSPs. 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. This similarity provides strong evidence that the complexity transitions are characteristic of the problem rather than the algorithm or representation of the problem.

The experiments with generalized, sine-based CSPs appear to show that in general various problem properties need to be taken into account to characterize problem complexity. In particular, varying the constraint tightness seems to be important when generating and analyzing problems.

More work is need to be able to explain the behaviors indicated in Figures 11 and 12. It is unclear whether the curves show phase transitions or continuously increasing slopes. In contrast to the SAT-like CSPs, these curves seem to exhibit super-exponential behavior. Also, computational cost of solving discrete SAT problems has a phase transition of easy-hard-easy. Feasible SAT problems with large constraint ratios tend to be easier to solve by both complete and local search solvers than critically constrained problems. Thus, we plan to run additional experiments for large constraint ratios on the continuous problems.

As a further next step, we are studying the behavior of constraint problems arising directly from the modular robot control problem of Section 2. Figure 13 shows early results for a set of sample problems similar to the one illustrated in Figure 4, with varying numbers of links and obstacles (i.e., constraints). The CSPs with 10 and 25 variables were solved by Solver 1 for various constraint ratios. The curves suggest a phase transition of problem complexity and possibly a similar convergence of iteration and restart curves as observed above, but a much deeper analysis of this problem is needed. We also intend to link these results back to the sine-based CSPs presented in this paper. Also, we will eventually extend this work to optimization 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. For larger problems, it becomes necessary to look for different solution methods, such as hierarchical and distributed solving approaches.

References

[Agrawal and Faiz, 1998] S. K. Agrawal and N. Faiz. Optimization of a class of nonlinear dynamic systems: new efficient method without lagrange multipliers. *J. of Optimization Theory and Application*, 97(1):11–28, April 1998.

[Agrawal and Garimella, 1994] S. K. Agrawal and R. Garimella. Workspace boundaries of free-floating open and closed chain planar manipulators. *J. of Mechanical Design*, 116:105–110, March 1994.

[Baillieul, 1986] J. Baillieul. Avoiding obstacles and resolving kinematic redundancy. In *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, pages 1698–1704, 1986.

[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.

[Chang and Wah, 1995] Y. Chang and B. W. Wah. Lagrangian techniques for solving a class of zero-one integer linear programs. In *Proc. Computer Software and Applications Conf.*, 1995.

[Chirikjian and Burdick, 1995] G. S. Chirikjian and J. W. Burdick. Kinematically optimal hyper-redundant manipulator configurations. *IEEE Trans. On Robotics and Automation*, 11:794–806, 1995.

[Chiu, 1988] S. L. Chiu. Task compatibility of manipulator postures. *Int. J. of Robotics Research*, 7(5):13–21, 1988.

[Craig, 1989] J. J. Craig. *Introduction to Robotics Mechanics and Control*. Addison Wesley, 1989.

[CUTE, 2001] CUTE. Constrained and unconstrained testing environment, 2001. http://www.cse.clrc.ac.uk/Activity/CUTE.

[Fromherz *et al.*, 1999] M. P. J. Fromherz, M. Hoeberechts, and W. B. Jackson. Towards constraint-based actuation allocation for hyper-redundant manipulators. In *CP'99 Workshop on Constraints in Control (CC'99)*, Alexandria, VA, 1999. See also http://www.parc.xerox.com/fromherz.

[Ghosal and Roth, 1988] A. Ghosal and B. Roth. A new approach for kinematic resolution of redundancy. *Int. J. of Robotics Research*, 7(2):22–35, 1988.

[Gokce and Agrawal, 1999] A. Gokce and S. K. Agrawal. Mass center of planar mechanism using auxiliary parallelograms. *Trans. of the ASME*, 121:166–168, March 1999.

[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 *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.

[Khatib, 1987] O. Khatib. A unified approach to motion and force control in robotic manipulators: The operational space formulation. *IEEE J. on Robotics and Automation*, 3(1):43–53, 1987.

[Klein *et al.*, 1995] C. A. Klein, C. Chu-Jenq, and S. Ahmed. A new formulation of the extended jacobian method and its use in mapping algorithmic singularities for kinematically redundant manipulators. *IEEE Trans. On Robotics and Automation*, 11:50–55, 1995.

[Kotay *et al.*, 1998] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule. In *Proc. of the Conference on Robotics and Automation (ICRA98)*, page 424. IEEE, 1998.

[Michalewicz and Schoenauer, 1996] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[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.

[Rus and Vona, 1999] D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. In *Proc. of the Conference on Robotics and Automation (ICRA99)*. IEEE, 1999.

[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 *IJCAI-01 Workshop on Stochastic Search Algorithms*, Seattle, WA, 2001.

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

[Yim, 1994] M. Yim. Locomotion with a unit-modular reconfigurable robot. Ph.D. Thesis, Dept. of Mechanical Engineering, Stanford University, 1994.