Journal of Global Optimization **14:** 1–25, 1999.

Efficient and Adaptive Lagrange-Multiplier Methods for Nonlinear Continuous Global Optimization

BENJAMIN W. WAH and TAO WANG
Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: wah, wangtao@manip.crhc.uiuc.edu

(Received 11 September 1997; accepted 24 March 1998)

Abstract. Lagrangian methods are popular in solving continuous constrained optimization problems. In this paper, we address three important issues in applying Lagrangian methods to solve optimization problems with inequality constraints. First, we study methods to transform inequality constraints into equality constraints. An existing method, called the slack-variable method, adds a slack variable to each inequality constraint in order to transform it into an equality constraint. Its disadvantage is that when the search trajectory is inside a feasible region, some satisfied constraints may still pose some effect on the Lagrangian function, leading to possible oscillations and divergence when a local minimum lies on the boundary of the feasible region. To overcome this problem, we propose the *MaxQ* method that carries no effect on satisfied constraints. Hence, minimizing the Lagrangian function in a feasible region always leads to a local minimum of the objective function. We also study some strategies to speed up its convergence. Second, we study methods to improve the convergence speed of Lagrangian methods without affecting the solution quality. This is done by an adaptive-control strategy that dynamically adjusts the relative weights between the objective and the Lagrangian part, leading to better balance between the two and faster convergence. Third, we study a trace-based method to pull the search trajectory from one saddle point to another in a continuous fashion without restarts. This overcomes one of the problems in existing Lagrangian methods that converges only to one saddle point and requires random restarts to look for new saddle points, often missing good saddle points in the vicinity of saddle points already found. Finally, we describe a prototype *Novel* (Nonlinear Optimization via External Lead) that implements our proposed strategies and present improved solutions in solving a collection of benchmarks.

Key words: Adaptive weights, Convergence speed, Global search, Inequality constraints, Lagrange-multiplier method, Local search, Nonlinear continuous constrained optimization, Oscillations, Trace-based search method

1. Introduction

Many applications in engineering, decision science and operation research are formulated as optimization problems. These applications include digital signal processing, structure optimization, engineering design, neural-network learning, computer-aided design for VLSI, and chemical control processing. High-quality

solutions in these applications may have significant economical impacts, leading to lower implementation and maintenance costs while improving the quality of outputs.

The *constrained nonlinear global optimization problems* that we study take the following form

$$\begin{array}{ll} \textit{minimize} & f(X) \\ \textit{subject to} & g(X) \leqslant 0 \qquad X = (x_1, \ldots, x_n) \in R^n \\ & h(X) = 0 \end{array} \tag{1}$$

where $f(X)$ is an objective function, $g(X) = [g_1(X), \ldots, g_k(X)]^T$ is a set of k inequality constraints, and $h(X) = [h_1(X), \ldots, h_m(X)]^T$ is a set of m equality constraints. All $f(X)$, $g(X)$, and $h(X)$ are assumed to be (nonlinear) differentiable real-valued continuous functions.

Global minimization looks for a solution that satisfies all the constraints and is no larger than any other local minimum. This is a challenging problem as there may not be enough time to find a feasible solution, or even if a feasible solution is found, there is no way to show that it is minimal. In practice, one only seeks as many local minima as possible that satisfy the constraints, and pick the best local minimum. Active research in the past three decades has produced a variety of methods to find solutions to constrained nonconvex nonlinear continuous optimization problems [7, 9, 10, 15, 19, 29]. In general, they are divided into transformational and non-transformational methods.

Non-transformational approaches include discarding methods, back-to-feasible-region methods, and enumerative methods. Discarding methods [11, 15] drop solutions once they are found to be infeasible, while back-to-feasible-region methods [12] attempt to maintain feasibility by reflecting moves from boundaries if such moves go out of the current feasible region. Both of these methods have been combined with global search and do not involve transformations to relax constraints. Last, enumerative methods [10] are generally too expensive to apply except for problems with linear objectives and constraints, and for bilinear programming problems [1].

Transformational approaches, on the other hand, transform a problem into another form before solving it. Some well-known methods include penalty, barrier, and Lagrange-multiplier methods [14]. *Penalty methods* transform constraints into part of the objective function and require tuning penalty coefficients either before or during the search. *Barrier methods* are similar except that barriers are set up to prevent solutions from going out of feasible regions. Both methods have difficulties when they start from an infeasible region or when feasible solutions are hard to find. However, they can be combined with other methods to improve their quality.

Lagrange-multiplier methods (or Lagrangian methods) introduce Lagrange variables to gradually resolve constraints through iterative updates. They are exact methods that optimize the objective using Lagrange multipliers to meet the Kuhn–Tucker conditions [14]. In view of their advantages, we use them for constraint

relaxation in this paper. Given an optimization problem with only equality constraints,

$$\begin{aligned} &\textit{minimize} \quad f(X) \\ &\textit{subject to} \quad h(X) = 0 \end{aligned} \tag{2}$$

the corresponding *Lagrangian function* and *augmented Lagrangian function* are defined as

$$\mathcal{L}(X, \lambda) = f(X) + \lambda^T h(X) \tag{3}$$

$$L(X, \lambda) = f(X) + \lambda^T h(X) + ||h(X)||_2^2 \tag{4}$$

where $\lambda = [\lambda_1, \ldots, \lambda_m]$ is a set of Lagrange multipliers. We use the augmented Lagrangian function in this paper as it provides better numerical stability.

According to classical optimization theory [14], all the extrema of (4) (called saddle points), whether local or global, are roots of the following set of first-order necessary conditions.

$$\nabla_X L(X, \lambda) = 0 \qquad \nabla_\lambda L(X, \lambda) = 0 \tag{5}$$

These conditions are necessary to guarantee the (local) optimality to the solution of (2). Because a saddle point is a local minimum of the Lagrangian function $L(X, \lambda)$ in the original-variable (X) space and a local maximum of $L(X, \lambda)$ in the Lagrange-multiplier (λ) space, it can be obtained by solving the following dynamic system of equations

$$\frac{d}{dt}X(t) = -\nabla_X L(X(t), \lambda(t)) \quad \frac{d}{dt}\lambda(t) = \nabla_\lambda L(X(t), \lambda(t)) \tag{6}$$

which perform descents in the original-variable space of X and ascents in the Lagrange-multiplier space of λ.

The goal of this paper is to develop new strategies to improve the convergence speed and solution quality of Lagrangian methods. We achieve this by three approaches. First, we study in Section 2 methods to eliminate oscillations when inequality constraints are transformed into equality constraints. These help improve convergence time as compared to existing methods of adding slack variables. Second, we present in Section 3 a method to improve the convergence speed of Lagrangian methods by adaptively adjusting the relative weights between the objective and the constraints. Third, we discuss in Section 4 a global-search method that looks for multiple saddle points in a continuous trajectory, without restarting the search from random starting points. We also present our prototype *Novel* that integrates these strategies. Finally, we show in Section 5 new improved results on a collection of benchmarks.

2. Handling inequality constraints

Lagrangian methods work well with equality constraints, but cannot directly deal with inequality constraints (1), except in some simple cases where one can directly

solve the first-order equations in closed form. In general, inequality constraints are first transformed into equivalent equality constraints before Lagrangian methods can be applied.

2.1. TRANSFORMATION USING SLACK VARIABLES

One possible transformation [14] to handle inequality constraint $g_i(X) \leqslant 0,\ i = 1, \cdots, k$, is to add a slack variable z_i to transform it into an equality constraint $g_i(X) + z_i^2 = 0$. After simplification [14], the augmented Lagrangian function for problem (1) becomes

$$L_z(X, \lambda, \mu) = f(x) + \lambda^T h(X) + ||h(X)||_2^2 + \sum_{i=1}^{k} \left[\max^2(0, \mu_i + g_i(X)) - \mu_i^2\right] \tag{7}$$

where λ and μ are Lagrange multipliers. In the same way as (6), a saddle point to (7) can be reached by doing descents in the X space and ascents in the λ and μ space.

The balance between descents and ascents depends on the magnitudes of Lagrange multipliers λ and μ, which play a role in balancing objective $f(X)$ and constraints $h(X)$ and $g(X)$ and in controlling indirectly the convergence speed and solution quality of the Lagrangian method. At a saddle point, the forces due to descent and ascent reach a balance through appropriate Lagrange-multiplier values.

To emphasize how the relative weights affect the convergence speed and solution quality, we introduce an additional weight w into (7) and get

$$L_0(X, \lambda, \mu) = w\ f(x) + \lambda^T h(X) + ||h(X)||_2^2 + \sum_{i=1}^{k} \left[\max^2(0, \mu_i + g_i(X)) - \mu_i^2\right] \tag{8}$$

where $w > 0$ is a weight on the objective. When $w = 1$, $L_0(X, \lambda, \mu) = L_z(X, \lambda, \mu)$, which is the original Lagrangian function. The corresponding dynamic system is as follows:

$$\frac{d}{dt}X(t) = -\nabla_X L_0(X(t), \lambda(t), \mu(t)) \tag{9}$$

$$\frac{d}{dt}\lambda(t) = \nabla_\lambda L_0(X(t), \lambda(t), \mu(t)) \tag{10}$$

$$\frac{d}{dt}\mu(t) = \nabla_\mu L_0(X(t), \lambda(t), \mu(t)) \tag{11}$$

Starting from an initial point $(X(0), \lambda(0), \mu(0))$, we solve the dynamic equations (9)–(11) using an ordinary differential equation solver *LSODE*[⋆] and observe a

[⋆] *LSODE* is a solver for first-order ordinary differential equations, a public-domain package available from http://www.netlib.org.

search trajectory $(X(t), \lambda(t), \mu(t))$. When a saddle point is on the boundary of the feasible region (which is true for most problems in the benchmark collection [6]), the dynamic equation approaches it from both the inside and outside of the feasible region. We observe three behaviors of the search trajectory:

- The trajectory gradually reduces its oscillations and eventually converges;
- The trajectory oscillates within some range but never converges;
- The magnitude of oscillations increases, and the trajectory eventually diverges.

To illustrate these three behaviors (divergence, oscillations without convergence, and reduction of oscillations until convergence), we apply (9)–(11) to solve Problem 2.3 in [6], which is given as follows:

$$\begin{aligned}
\textit{Minimize}\quad & 5x_2 + 5x_3 + 5x_4 + 5x_5 - x_6 - x_7 - x_8 - x_9 - x_{10} - 8x_{11} - 8x_{12} \\
& - 8x_{13} - x_{14} - 5x_2^2 - 5x_3^2 - 5x_4^2 - 5x_5^2 \qquad (12)\\
\textit{subject to}\quad & 2x_2 + 2x_3 + 8x_{11} + 8x_{12} \leqslant 10.0, \qquad 0 \leqslant x_i \leqslant 1.0, \qquad i = 2, 3, \ldots, 14,\\
& 2x_2 + 2x_4 + 8x_{11} + 8x_{13} \leqslant 10.0, \qquad 2x_3 + 2x_4 + 8x_{12} + 8x_{13} \leqslant 10.0,\\
& 8x_{11} - 8x_2 \leqslant 0.0, \qquad 8x_{12} - 8x_3 \leqslant 0.0,\\
& 8x_{13} - 8x_4 \leqslant 0.0, \qquad -2x_5 - x_6 + 8x_{11} \leqslant 0.0,\\
& -2x_7 - x_8 + 8x_{12} \leqslant 0.0, \qquad -2x_9 - x_{10} + 8x_{13} \leqslant 0.0,
\end{aligned}$$

It is a quadratic nonlinear programming problem with linear inequality constraints. We set $X(t = 0)$, the initial point at $t = 0$, at the middle of the search space, i.e., $x_i = 0.5, i = 2, 3, \ldots, 14$ and $\lambda(t = 0) = \mu(t = 0) = 0$. We further set the maximum (logical) time for *LSODE* to be $t_{\max} = 10^5$, which is divided into small units of $\triangle t = 1.0$, resulting in a maximum of 10^5 iterations $(= t_{\max}/\triangle t)$. The stopping condition for (9)–(11) is the Lyupunov condition:

$$||dX(t)/dt||^2 + ||d\lambda(t)/dt||^2 + ||d\mu(t)/dt||^2 \leqslant \delta = 10^{-25} \qquad (13)$$

The dynamic system stops when it converges or when it reaches the maximum number of iterations.

When $w = 1$, (9)–(11) diverge quickly into infinity, meaning that the original Lagrangian method will diverge. If we scale the objective by 10 (i.e., $w = 1/10$), then the objective value $f(X(t))$ oscillates within the range $[-17, -10]$, while the maximum violation $v_{\max}(t)$ is between 0 and 0.4, as shown in Figure 1. Here, $v_{\max}(t)$ at time t is defined as

$$v_{\max}(t) = \max_{1 \leqslant i \leqslant m, 1 \leqslant j \leqslant k} \{|h_i(X(t))|, \max\left[0, g_j(X(t))\right]\} \qquad (14)$$

If we further reduce w to $1/15$, then the oscillations subside, and the trajectory eventually converges (see Figure 2).

Intuitively, the occurrence of oscillations can be explained as follows. Suppose we start from an infeasible point initially $(t = 0)$ where inequality constraint $g_i(X(t = 0))$ is violated; i.e., $g_i(X(t = 0)) > 0$. As the search progresses, the

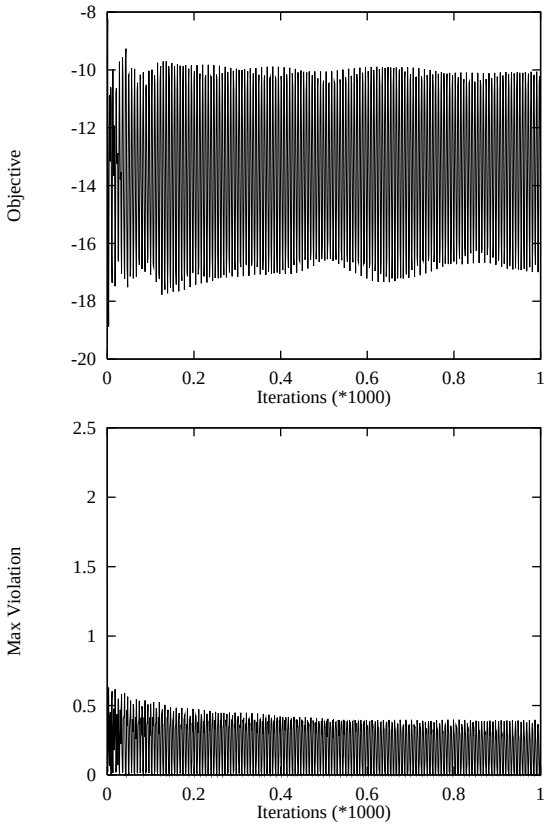

Figure 1. The objective and maximum violation oscillate using the slack-variable method ($w = \frac{1}{10}$).

corresponding $\mu_i(t)$ increases and pushes the trajectory towards a feasible region. At some time t, the inequality constraint $g_i(X(t)) \leqslant 0$ is satisfied for the current point $X(t)$. At this point, $d\mu_i(t)/dt = \max(0, g_i(X) + \mu_i) - \mu_i$, and is negative when $g_i(X) < 0$. Hence, the trajectory decelerates but continues to move into the feasible region even when the corresponding constraint, $g_i(X(t))$, is satisfied. The movement of the trajectory inside the feasible region eventually stops because the local minimum is on the boundary, and the corresponding force due to descents in the objective space pushes the trajectory outside the feasible region. Likewise, when the trajectory is outside the feasible region, a force due to the constraints pushes the trajectory inside the feasible region. If these two forces are not well balanced, the search may diverge or oscillate without convergence.

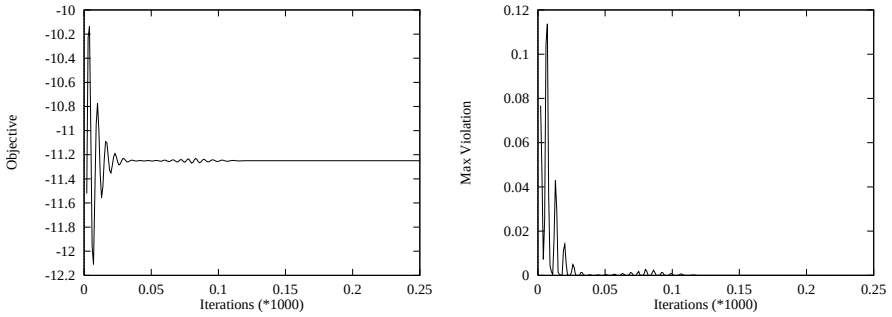

Figure 2. The objective and maximum violation converge after oscillations subside using the slack-variable method ($w = \frac{1}{15}$).

2.2. TRANSFORMATION USING THE MAXQ METHOD

To avoid oscillations in the method based on slack variables, we would like a search to converge to a local minimum without oscillations when it is on the boundary of a feasible region and when the trajectory is outside the feasible region. This is done by our proposed *MaxQ* method.

Without loss of generality, we ignore equality constraints in the following discussion for simplicity, knowing that equality constraints are handled in the way described in Section 1. The *MaxQ* method transforms an inequality constraint as follows.

$$\begin{aligned} &g_i(X) \leqslant 0 \quad \Longleftrightarrow \quad p_i(X) = [\max(0, g_i(X))]^{q_i} = 0 \\ &q_i > 1, \quad i = 1, \ldots, k \end{aligned} \tag{15}$$

where q_i are control parameters to be determined later. For a given X, if $g_i(X) > 0$, $[\max(0, g_i(X))]^{q_i} = [g_i(X)]^{q_i}$; otherwise, $[\max(0, g_i(X))]^{q_i} = 0$. The augmented Lagrangian function is

$$\begin{aligned} L_q(X, \mu) &= f(X) + \mu^T p(X) + ||p(X)||_2^2 \\ &= f(X) + \sum_{i=1}^{k} \left[\mu_i p_i(X) + p_i^2(X)\right] \end{aligned} \tag{16}$$

where $p(X) = [p_1(X), p_2(X), \ldots, p_k(X)]^T$.

It is important to choose suitable control parameters q_i $(i = 1, \ldots, k)$ because they affect the convergence speed of our method. One can easily show that, when $q_i \geqslant 1$, inequality constraint $g_i(X) \leqslant 0$ is equivalent to equality constraint $p_i(X) = \max^{q_i}(0, g_i(X)) = 0$. Suppose that q_i is a constant. Using a dynamic system similar to that in (9)–(11) to solve Lagrangian function (16), we need to evaluate partial derivative $\nabla_X p_i(X)$, where

$$\nabla_X p_i(X) = q_i[\max(0, g_i(X))]^{q_i - 1} \nabla_X g_i(X) = p_i'(X) \nabla_X g_i(X)$$

If $q_i \leqslant 1$, $p_i'(X)$ is not continuous, and the derivative of $L_q(X, \mu)$ is not continuous when $g_i(X) = 0$. However, the continuity of derivatives is required by most differential-equation solvers, such as *LSODE*. For this reason, we require $q_i > 1$ in (15).

One way of selecting q_i is to make it very close to 1, namely, $q_i \to 1$. At this point, the dynamic system will approach a feasible region slowly when the saddle point is on the boundary of the feasible region. This is true because $p_i'(X) \simeq 1$ if $g_i(X) > 0$, independent of how far the current point X is away from the feasible region. Thus, larger control parameters q_i are needed for fast convergence if the current point X is far from the feasible region. In contrast, if we choose $q_i \gg 1$, then $p_i'(X) \simeq 0$ as $g_i(X) \to 0$, meaning that *LSODE* converges very slowly towards the saddle point on the boundary of the feasible region.

Taking these facts into account, in order to have fast convergence, we should adapt q_i dynamically as the search goes to a saddle point. Since different inequality constraints may have different convergence rates to the saddle point, we associate inequality constraint $g_i(X) \leqslant 0$ with its own control parameter q_i. Each parameter will be updated dynamically based on the value of $g_i(X)$: q_i is large if $g_i(X) \gg 0$, and q_i is gradually reduced to a value approaching 1 when the search is close to the saddle point on the boundary. One possible choice of q_i is as follows:

$$q_i(g_i(X)) = \frac{s_0}{1 + \exp(-s_1 g_i(X))} \tag{17}$$

where $s_0 = 2$ and $s_1 > 0$ are two parameters that control the shape of function $q_i(x)$. When $g_i(X)$ approaches 0, q_i will approach 1. The dynamic equation to solve (16) is

$$\begin{aligned} \frac{d}{dt}X(t) =& - \nabla_X L_q(X(t), \mu(t)) = - \nabla_X f(X) \\ & - \sum_{i=1}^{k} (\mu_i + 2p_i(X)) \nabla_X p_i(X) \end{aligned} \tag{18}$$

$$\frac{d}{dt}\mu_i(t) = \nabla_\mu L_q(X(t), \mu(t)) = p_i(X) \tag{19}$$

where

$$\begin{aligned} \nabla_X p_i(X) = \big[& q_i'(g_i(X)) p_i(X) \log_e \max(0, g_i(X)) \\ & + q_i(g_i(X)) [\max(0, g_i(X))]^{q_i(g_i(X))-1} \big] \nabla_X g_i(X) \end{aligned} \tag{20}$$

In the proof above, we assume that q_i takes the form in (17), where $s_0 = 2$. With this choice, $\nabla_X p_i(X)$ changes very fast from 1 to 0 near the saddle point as $g_i(X) \longrightarrow 0$, making it difficult for *LSODE* to find a suitable step size in order to reach the saddle point. To let the gradient change smoothly, we set $s_0 = 2.5$ or $s_0 = 3.0$, and set s_1 to satisfy $q_i(g_i(X)) = 2$ when $g_i(X) = 1$. Thus, $s_1 = -\log_e[s_0/2 - 1]$.

Note that (16) is similar to (7) in the sense that both use the max function. The main difference is that (16) avoids the case in (7) in which inequality constraint $g_i(X(t)) \leqslant 0$ is satisfied at time t but $g_i(X(t))$ also appears in the Lagrangian function. When $g_i(X(t))$ is satisfied, it is meaningful to minimize $f(X)$ independent of the value of $g_i(X(t))$.

2.3. CONVERGENCE PROPERTIES OF THE MAXQ METHOD

There are two types of saddle points X^* as shown in Figure 3.When saddle point X^* is within a feasible region, i.e., $g_i(X^*) < 0$ (see Figure 3a), $p_i(X^*) = 0$, and $\nabla_X f(X^*) = 0$. This means that X^*, an equilibrium point of dynamic system (18), (19), is given by

$$\frac{d}{dt}X(t) = 0 \quad \text{and} \quad \frac{d}{dt}\mu_i(t) = 0 \qquad i = 1, \ldots, k \tag{21}$$

Thus, the trajectory controlled by (18), (19) will converge to this saddle point X^*.

When the saddle point X^* is on the boundary of the feasible region shown in Figure 3b, it will be asymptotically approached from outside the feasible region (right side of Figure 3b). In order to prove this, we only need to show that X^* is asymptotically a regular point of the constraints $p_i(X) = 0$ [14] because inequality constraint $g_i(X) \leqslant 0$ has been transformed into an equivalent equality constraint $p_i(X) = 0$.

Because X^* is on the boundary of the feasible region, $g_i(X^*) = 0$, and so $p_i(X^*) = 0$. In addition, when $X \longrightarrow X^*$, $g_i(X) \longrightarrow g_i(X^*) = 0$, and $q_i(g_i(X)) \longrightarrow q_i(g_i(X^*)) = 1$. By taking limits, we obtain

$$\lim_{X \to X^*} q_i'(g_i(X))p_i(X)\log_e \max(0, g_i(X)) = 0 \tag{22}$$

$$\lim_{X \to X^*} q_i(g_i(X))[\max(0, g_i(X))]^{q_i(g_i(X))-1} = 1 \tag{23}$$

Therefore, $\lim_{X \to X^*} \nabla_X p_i(X) = \nabla_X g_i(X^*)$ according to (20), which means that asymptotically regular point X^* of $p_i(X)$ is the same as that of the original con-

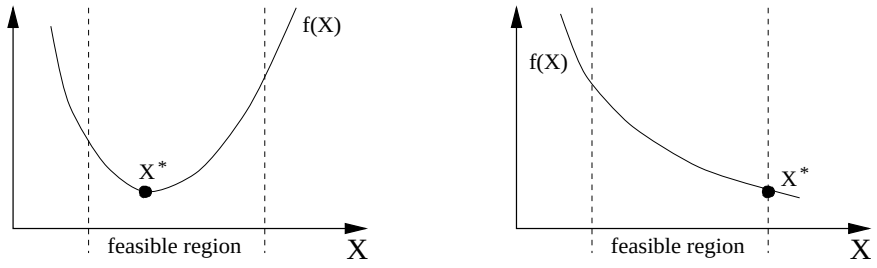

Figure 3. Relationship between saddle point and feasible region. (a) The saddle point is within the feasible region; (b) the saddle point is on the boundary of feasible region.

straint $g_i(X) \leqslant 0$. Since a saddle point, if it exists, must be a regular point of $g_i(X)$ [14], X^* can be asymptotically reached by the dynamic system (18), (19).

2.4. DYNAMIC CONVERSION OF INEQUALITY CONSTRAINTS TO EQUALITY ONES IN MAXQ

As discussed above, if solution X^* is on the boundary of a feasible region, i.e., when some $g_i(X^*)$ equals zero, then dynamic system (18), (19) cannot reach this point exactly. This point can only be approached with a precision proportional to the convergence condition of the *LSODE* solver. However, this may take a long time even if q_i is changed dynamically. This happens because the violation of constraint $g_i(X^*)$ is small when the current point $X(t)$ is close to the boundary X^* (from the right side of Figure 3b). Consequently, the increment of the corresponding Lagrange multiplier μ_i will be small, leading to slow progress towards the saddle point.

Suppose we know that some $g_j(X^*) = 0$ for a given solution. In this case, faster convergence can be achieved if we consider $g_j(X)$ as an equality constraint, $g_j(X) = 0$, rather than an inequality constraint, $g_j(X) \leqslant 0$. The difficulty, however, is that it is impossible to know in advance which inequality constraints $g_j(X) \leqslant 0$ will satisfy the boundary condition (i.e., $g_j(X) = 0$) for a solution on the boundary.

We can utilize the property that, when $X(t)$ is very close to the boundary with respect to a particular inequality constraint ($g_j(X(t))$ is positive and close to zero) and convergence of this constraint is slow, it is very likely that the saddle point is on the boundary with respect to this constraint. At this point, we can dynamically convert inequality constraint $g_j(X(t)) \leqslant 0$ into equality constraint $g_j(X(t)) = 0$ to improve the convergence rate.

Since we solve the dynamic system using *LSODE*, let X and X_0 be the points of two successive iterations. The conversion of inequality constraint $g_j(X) \leqslant 0$ occurs when the following two conditions are satisfied.

- The dynamic system converges to some point X when it changes very little for a window of 20 iterations. We define the current point to have slow convergence when $\#\{\max_i |x_i - x_{0i}| / \max_j |x_j| < \delta\} \geqslant 10$ ($\delta = 10^{-4}$ in our experiments).
- The dynamic system converges to the boundary when $g_j(X)$ is very close to zero; that is, $0 < g_j(X) < \epsilon$ ($\epsilon = 10^{-4}$ in our experiments).

Both conditions are very important. Without the first condition, trajectory $X(t)$ that occasionally passes the boundary of $g_j(X(t))$ would erroneously cause some inequality constraints to be converted. It makes sure that the trajectory really changes very little during a period of time. The second condition guarantees that only those inequality constraints very close to the boundary can be converted into equality ones. Note that dynamic conversion can happen to many inequality constraints at the same time as long as they satisfy these two conditions.

If dynamic conversion is performed on inequality constraint $g_j(X) \leqslant 0$, then the terrain of Lagrangian function $L_q(X, \mu)$ will be changed and be totally different. To maintain the search direction in the original-variable space X, we have to adjust Lagrange multiplier μ_j. Let the current point just before the conversion be (X, μ_j). The Lagrangian term associated with inequality constraint $g_j(X) \leqslant 0$ is

$$\begin{aligned} L_j(X, \mu_j) &= \mu_j [\max(0, g_j(X))]^{q_j} + [\max(0, g_j(X))]^{2q_j} \\ &= \mu_j g_j^{q_j}(X) + g_j^{2q_j}(X) \end{aligned}$$

according to the conversion conditions. The derivative of $L_j(X, \mu_j)$ with respect to X and μ_j are

$$\begin{aligned} &\nabla_X L_j(X, \mu_j) = \\ &\quad = \left[\mu_j g_j^{q_j-1}(X) + 2g_j^{2q_j-1}(X)\right]\left[q_j + q_j'(X) g_j(X) \log_e g_j(X)\right] \nabla_X g_j(X) \\ &\nabla_{\mu_j} L_j(X, \mu_j) = g_j^{q_j}(X) \end{aligned}$$

Let $(X, \hat{\mu}_j)$ be the point after the conversion. This means that we apply equality constraint $g_j(X) = 0$ at the current point $(X(t), \hat{\mu}_j)$ where inequality constraint $g_j(X) \leqslant 0$ was before. The Lagrangian term related to $g_j(X) = 0$ is

$$\hat{L}_j(X, \hat{\mu}_j) = \hat{\mu}_j g_j(X) + g_j^2(X)$$

and the derivative of $\hat{L}_j(X, \hat{\mu}_j)$ with respect to X and $\hat{\mu}_j$ are

$$\begin{aligned} &\nabla_X \hat{L}_j(X, \hat{\mu}_j) = (\hat{\mu}_j + 2g_j(X)) \nabla_X g_j(X) \\ &\nabla_{\hat{\mu}_j} \hat{L}_j(X, \hat{\mu}_j) = g_j(X) \end{aligned}$$

Since the control parameter q_j is close to 1, the search direction in the Lagrange-multiplier space changes very little, meaning that $\nabla_{\mu_j} L_j(X, \mu_j) \simeq \nabla_{\hat{\mu}_j} \hat{L}_j(X, \hat{\mu}_j)$, independent of the value μ_j. To retain the search direction in the original-variable space X, we set $\nabla_X L_j(X, \mu_j) = \nabla_X \hat{L}_j(X, \hat{\mu}_j)$ and get

$$\hat{\mu}_j = \left[\mu_j g_j^{q_j-1}(X) + 2g_j^{2q_j-1}(X)\right]\left[q_j + q_j'(X) g_j(X) \log_e g_j(X)\right] - 2g_j(X) \tag{24}$$

2.5. ILLUSTRATION OF THE MAXQ METHOD

As is in the slack-variable method, we introduce an additional weight w into the original augmented Lagrangian function in the *MaxQ* method.

$$L_m(X, \lambda, \mu) = w\ f(x) + \lambda^T h(X) + ||h(X)||_2^2 + \mu^T p(X) + ||p(X)||_2^2 \tag{25}$$

where $w > 0$ is a weight on the objective, λ is the Lagrange multiplier for equality constraints, μ, the Lagrange multiplier for inequality constraints, and $p(X) = [p_1(X), p_2(X), \ldots, p_k(X)]^T$.

To show how *MaxQ* avoids divergence and oscillations that occur in the slack-variable method, we consider the same Problem 2.3 defined in (12) [6]. The starting point is at the middle of the search space, the same as that used in the slack-variable method. Three cases were tested: no scaling, scaling the objective by 10 (i.e., $w = 1/10$), and scaling the objective by 15. All of them converge with similar behavior. Figure 4 shows the second case. Obviously, *MaxQ* has smoother and better convergence property as compared to the slack-variable method.

The solution to Problem 2.3 is on the boundary of the feasible region as shown in Figure 3b. Since all the Lagrange multipliers are zero initially, only objective function $f(X)$ takes effect in the Lagrangian function, causing the trajectory to move away from the feasible region. The Lagrange multipliers then increase, pushing the trajectory back towards the boundary. Hence, the objective-function value increases, and the value of the maximum violation decreases (see Figure 4).

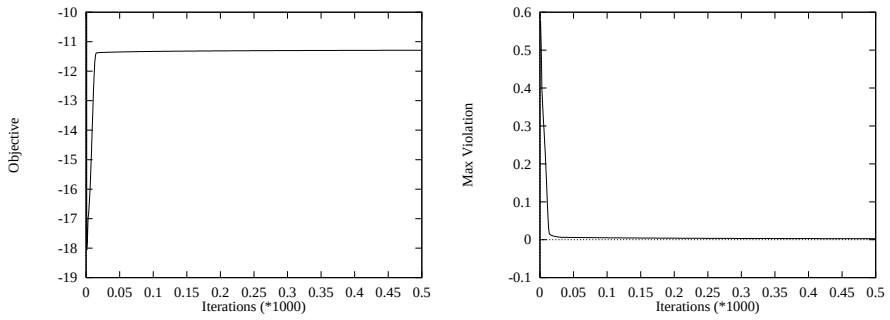

Figure 4. The objective and maximum violation converge smoothly using *MaxQ* for Problem 2.3 defined in (12) [6] ($w = \frac{1}{10}$). The number of iterations required is large and is over 94,000.

Note that there is a gap between our current implementation of *MaxQ* and its theoretic result, in the sense that the analytic proof requires $s_0 = 2$ in (17) but *LSODE* uses $s_0 = 2.5$ or 3.0. This gap causes the *MaxQ* method to converge slowly sometimes, like the case shown in Figure 4 that requires over 94,000 iterations. In this example, *MaxQ* reduces the constraint violation very quickly in the beginning, but slowly afterwards.

This problem can be solved by two approaches. The first is to use another differential-equation solver that is insensitive to quick changes of gradients, making the analytic result hold during the search. This will be investigated in the future. The second is to adaptively adjust the relative weight w between the objective and the constraints during the search. As soon as we detect divergence, oscillations, or slow convergence, w is adjusted accordingly. When we apply this strategy in the slack-variable method [34], it is able to eliminate divergence, reduce oscillations,

and greatly speed up convergence. We discuss the application of this strategy in the *MaxQ* method in the next section.

3. Adaptive Lagrange-multiplier method

In the last section, we have studied two methods to handle inequality constraints. We have shown that the slack-variable method is sensitive to the relative weights between the objective and the constraints, and that its implementation may diverge, oscillate or converge. Although the *MaxQ* method does not oscillate and is not sensitive to the relative weights, careful weighting may help accelerate its convergence speed. The proper weights for fast convergence, however, are problem-dependent and are impossible to choose in advance. In this section, we describe a strategy for *MaxQ* to dynamically adapt the weights based on the behavior of the search progress. This is based on the strategy we have developed for the slack-variable method [34].

3.1. DYNAMIC WEIGHT-ADAPTATION STRATEGY

Lagrangian methods rely on two counteracting forces to resolve constraints and find high-quality solutions. When all the constraints are satisfied, Lagrangian methods rely on gradient descents in the objective space to find local minima. On the other hand, when any of the constraints are not satisfied, Lagrangian methods rely on gradient ascents in the Lagrange-multiplier space in order to increase the penalties on unsatisfied constraints and to force the constraints into satisfaction. The balance between gradient descents and gradient ascents depends on the magnitudes of Lagrange multipliers λ and μ, which play a role in balancing objective $f(X)$ and constraints $h(X)$ and $g(X)$ and in controlling indirectly the convergence speed and solution quality of the Lagrangian method. At a saddle point, the forces due to descent and ascent reach a balance through appropriate Lagrange-multiplier values.

Combining augmented Lagrangian functions (8) and (25), we get a general augmented Lagrangian function as follows:

$$L_b(X, \lambda, \mu) = w\ f(x) + \lambda^T h(X) + ||h(X)||_2^2 + L_{ineq}(\mu, X) \tag{26}$$

where w is the relative weight, λ is the Lagrange multiplier for equality constraints, and μ for inequality constraints. $L_{ineq}(\mu, X)$ depends on the way to deal with inequality constraints. It can be the slack-variable method or *MaxQ*. Starting from an initial point $(X(0), \lambda(0), \mu(0))$, the dynamic system to find saddle points is

$$\frac{d}{dt}X(t) = -\nabla_X L_b(X(t), \lambda(t), \mu(t)) \tag{27}$$

$$\frac{d}{dt}\lambda(t) = \nabla_\lambda L_b(X(t), \lambda(t), \mu(t)) \tag{28}$$

$$\frac{d}{dt}\mu(t) = \nabla_\mu L_b(X(t), \lambda(t), \mu(t)) \tag{29}$$

```
1. Set control parameters:
       time interval △t
       initial weight w(t = 0)
       maximum number of iterations i_max
2. Set window size N_w = 100 or 10
3. j := 1          /* j is the iteration number */
4. while j ⩽ i_max and stopping condition is not satisfied do
5.     advance search trajectory by △t in LSODE to get to (X_j, λ_j, μ_j)
6.     if trajectory diverges then
           reduce w; restart the algorithm by going to Step 2
       end if
7.     record useful information for calculating performance metrics
8.     if ((j mod N_w) == 0) then
           /* Test whether w should be modified at the end of a window */
9.         compute performance metrics based on data collected
10.        change w when the conditions are satisfied
       end if
11. end while
```

Figure 5. Framework of the dynamic weight-adaptation algorithm.

Figure 5 outlines the algorithm. Its basic idea is to first estimate the initial weight $w(t = 0)$ (Step 1), measure the performance metrics of the search trajectory $(X(t), \lambda(t), \mu(t))$ periodically, and adapt $w(t)$ to improve convergence time or solution quality.

Let $t_{\max}$ be the total (logical) time for the search, and $t_{\max}$ be divided into small units of time $\triangle t$ so that the maximum number of iterations $i_{\max} = t_{\max}/\triangle t$. Further, assume a stopping condition if the search were to stop before $t_{\max}$ (Step 4). Given a starting point $X(t = 0)$, we set the initial values of the Lagrange multipliers to be zero; i.e., $\lambda(t = 0) = \mu(t = 0) = 0$. Let (X_i, λ_i, μ_i) be the point of the ith iteration, and $v_{\max}(i)$ be its maximum violation defined in (14).

To monitor the progress of the search trajectory, we divide time into non-overlapping windows of size N_w iterations each (Step 2). In each window, we compute some metrics to measure the progress of the search relative to that of previous windows. For the qth window ($q = 1, 2, \ldots$), we calculate the average value of $v_{\max}(t)$ using (14) over all the iterations in this window,

$$\bar{v}_q = \frac{1}{N_w} \sum_{t=(q-1)N_w+1}^{mN_w} v_{\max}(t) \tag{30}$$

During the search, we employ *LSODE* to solve dynamic system (27)–(29), and advance the trajectory by time interval $\triangle t$ in each iteration in order to arrive at point (X_j, λ_j, μ_j) (Step 5).

At this point, we test whether the trajectory diverges or not (Step 6). Divergence happens when the maximum violation $v_{\max}(t)$ is larger than an extremely large

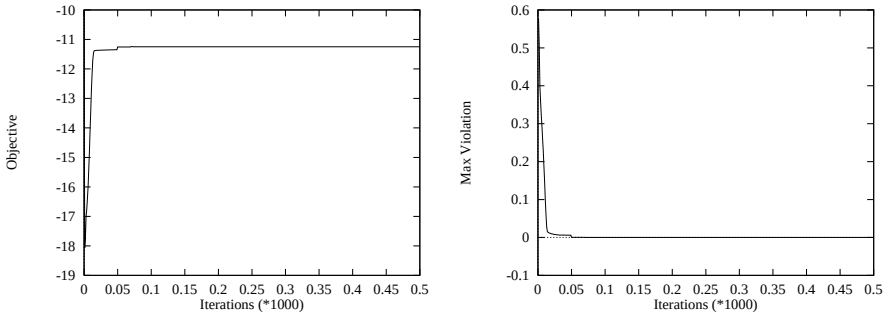

Figure 6. The objective and maximum violation converge after 756 iterations for *MaxQ* using dynamic weight adaptation (initial $w = \frac{1}{10}$).

value (e.g. 10^{20}). If it happens, we reduce w by a large amount, say $w \Longleftarrow w/10$, and restart the algorithm. In each iteration, we also record some statistics such as $v_{\max}(t)$, which will be used to calculate the performance metrics for each window (Step 7).

At the end of each window or every N_w iterations (Step 8), we decide whether to update w based on the performance metrics (30) (Step 9). In our current implementation, we use the average value of maximum violation $v_{\max}(t)$. In general, other application-specific metrics can also be used. Based on these measurements, we adjust w accordingly (Step 10).

As explained in Section 2.5, the major problem with *MaxQ* sometimes is its slow convergence, which can be measured by how fast the maximum violation decreases. Therefore, we monitor the reduction of $\bar{v}_q$, the average value of the maximum violation. If it is found to decrease slowly, i.e., $\bar{v}_{q-1} - \bar{v}_q \leqslant \beta \bar{v}_{q-1}$ where β is a threshold (e.g. $\beta = 10\%$), we will reduce weight w by half ($w \Longleftarrow w/2$). Its effect is to put more weight on the constraints, thus pushing the trajectory quickly to a feasible region. Note that when comparing the values between two successive windows $q-1$ and q, both must use the same weight w; otherwise, the comparison is not meaningful because the terrain may be different. Hence, after adapting w, we should wait at least two windows before changing it again.

3.2. ILLUSTRATION OF THE WEIGHT-ADAPTATION STRATEGY FOR MAXQ

To use the dynamic weight-adaptation method, we set time interval $\triangle t = 1$ for *LSODE*, and window size $N_w = 10$. Corresponding to Figure 4, we start from the initial weight $w(t = 0) = 1/10$ and the same starting point $(X(0), \lambda(0), \mu(0))$. Figure 6 shows the resulting search profile, in which the search converges using only 756 iterations, which is significantly better than the 94,000 iterations without weight adaptation.

It is important to note that solution quality is the same as that in Figure 4, and both obtain the objective value -11.25 when the search converges.

The remaining issue in our algorithm is to choose a good starting value for $w(t = 0)$. If $w(0)$ is too large, then it is difficult for the constraints to be satisfied, resulting in slow convergence for *MaxQ*. If $w(0)$ is too small, then the objective is under-weighted, and the search may converge to a worse saddle point. After studying many benchmark problems [6], we found that if we set the starting points of the search to be the ones given by [6] and $w(0) = 1$, then both the slack-variable and *MaxQ* methods work well. This suggests us to start with $w(0) = 1$ in our experiments. A small number of the problems may still need further adaptation of w during the search.

4. Global search for saddle points

The Lagrangian method presented in the last two sections only looks for a single saddle point, behaving like a local search. To find multiple saddle points, *global-search methods* are needed to bring the search out of local saddle points. In this section, we present our global search strategy, followed by a description of our trace-based search method. We first describe some previous work on solving unconstrained nonlinear optimization problems since the dynamic system for solving a Lagrangian formulation can be treated as an unconstrained nonlinear optimization problem.

4.1. PREVIOUS WORK ON UNCONSTRAINED NONLINEAR OPTIMIZATION

Unconstrained nonlinear optimization, in general, is multi-modal with the following features. (a) Flat regions may mislead any gradient-based methods. (b) Gradients may differ by many orders of magnitude, making it difficult to use gradients in any gradient-based search method. (c) The existence of many local minima may trap some search methods, leading to suboptimal solutions. Based on these observations, *global search methods* should be able to use gradient information to descend into local minima, and be able to escape from local minima once it gets there.

Search methods can be classified into local and global. Local search algorithms, such as gradient-descent and Newton's methods, find local minima efficiently and work best in uni-modal problems. Global-search algorithms, in contrast, employ some heuristic or systematic strategies to look for global minima and do not stop after finding a local minimum [14, 18, 29]. Note that both gradients and Hessians can be used in local- and global-search methods [29].

Local-search algorithms have difficulty when the surface is flat where gradients are close to zero, or when gradients can be in a large range, or when the surface is very rugged. If gradients vary greatly, a search may progress too slowly when gradients are small or may over-shoot when gradients are large. If the function surface is rugged, a local search from a randomly chosen starting point will most likely converge to a local minimum that is near the starting point and results in a

solution worse than the global minimum. Moreover, these algorithms may require some properly chosen parameters as incorrectly chosen parameters may cause slow or unstable convergence.

To avoid getting stuck in local minima in local-search methods, many global-search methods have been developed. These methods rely on local search to determine local minima, but focus on bringing the search out of local minima when it gets there.

Few deterministic methods have been developed, most of which apply deterministic heuristics to bring a search out of a local minimum (such as those in covering and generalized descent methods) [10, 24, 29]. These deterministic methods do not work well when the search space is too large to be covered adequately [4, 9, 29]. Trajectory methods rely on an internal force (such as the momentum of the trajectory) to continue to move the trajectory once it gets to a local minimum. They do not work well when gradients are steep and the search space is rugged.

Probabilistic methods, on the other hand, rely on probability to make decisions. The simplest probabilistic algorithm uses restarts to bring a search out of a local minimum when little improvement can be made locally [20, 35, 36]. More advanced methods rely on probability to indicate whether the search should ascend from a local minimum. Simulated annealing is one of these methods that accepts up-hill movements [2, 3, 13, 21, 22, 30] based on some probability. Other stochastic methods rely on probability to decide which intermediate points to be interpolated as new starting points, such as random recombinations and mutations in evolutionary algorithms [8]. All these algorithms are weak in either their local search [14] or their global search [9, 23, 24, 29]. For instance, gradient information useful in local search is not well used in simulated annealing and evolutionary algorithms. In contrast, gradient-descent algorithms with multi-start are weak in their global search.

Other probabilistic methods utilize sampling to determine the terrain and to decide where to search [16, 24, 29, 31]. Such strategies may fail when the terrain is very rugged or when the search gets trapped in a deep but suboptimal basin. This happens in clustering methods whose performance is similar to that of random restarts when the terrain is rugged [24, 28]. Bayesian methods also do not work well. Most samples that they collect randomly from the error surface are close to the average error value, and these samples are inadequate to model the behavior at minimal points [17, 27, 29, 31]. In addition, they are computationally expensive and are usually not applicable for problems with over twenty variables.

4.2. *novel*: A TRACE-BASED SEARCH METHOD

To find saddle points of (26), Equation (27) performs descents in the original-variable space to locate local minima of the objective function when the constraints are satisfied, whereas (28), (29) perform ascents in the Lagrange-multiplier space when the constraints are violated.

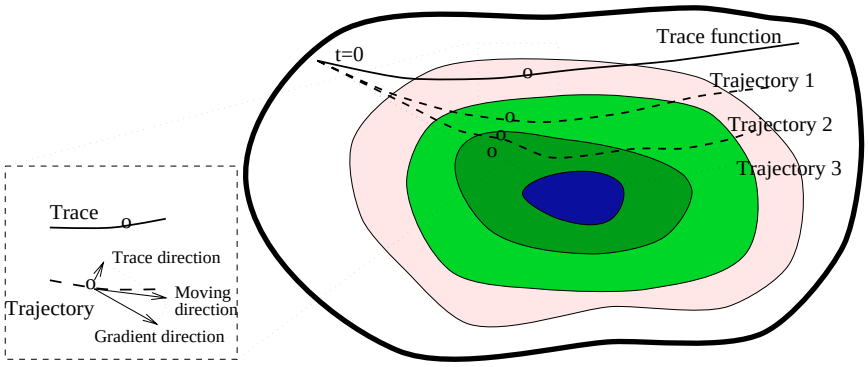

Figure 7. Novel has two phases: global search and local refinement. In the global-search phase, the trajectory combines a Lagrangian search and the pull exerted by the moving trace. In the local-search phase, the trajectory is sampled to collect starting points for pure Lagrangian searches.

We are interested to move the trajectory from one saddle point in a feasible space to another, without having to restart the search. To do so, we add an *external* force to pull the search out of a saddle point in the original-variable space continuously and escapes from it without restarts [26, 32]. It has three features: exploring the solution space, locating promising regions, and finding saddle points. In exploring the solution space, the search is guided by a continuous terrain-independent trace that does not get trapped by local saddle points. This trace is usually an aperiodic continuous function that runs in a bounded space. It continues to move over the search space independent of local gradients. In locating promising regions, our trace-based method uses local Lagrangian search to attract the search to a saddle point but relies on the trace to pull it out once little improvement can be found. Finally, our trace-based method selects one initial point from each promising local region and uses them as initial points for a Lagrangian search to find saddle points.

In exploring the search space, the trace plays an important role in discovering regions with new local saddle points. A *trace* is a continuous aperiodic function of (logical) time that generates a *trajectory*. At time $t = 0$ both the trace and the trajectory start at the same point. As the trace moves from point X_1 to point X_2, the trajectory moves from point Y_1 to Y_2, where Y_2 is a function of the local gradient at Y_1 and the distance between X_1 and Y_1 (see Figure 7). These two forces (descent into a local saddle point and attraction exerted by the trace) form a composite vector that represents the route taken by the trajectory.

When dealing with constrained problems formulated by Lagrangian functions, there are two different sets of variables, the original variables X and the Lagrange multipliers λ and μ. Intuitively, there is no need of a trace in the Lagrangian space because the Lagrange multipliers are responsible to bring the trajectory to a feasible region and does not involve finding local minima. As the trace pulls the search out of a local saddle point and enters an infeasible region, the corresponding Lagrange

multipliers will be automatically increased based on the dynamic system, and then push the trajectory back into the feasible region. In this sense, the Lagrange multipliers are passive since they change with the constraints. If one also uses a trace function in the Lagrangian space, the force imposed by the trace and that by the Lagrange multipliers may contradict. In short, the trace does not affect the search in the Lagrange-multiplier space defined by (28) and (29). This strategy corrects a problem in our original trace-based strategy presented in [32] that uses a trace in the Lagrangian space.

The overall dynamic system for nonlinear constrained optimization is as follows.

$$\frac{d}{dt}X(t) = -\mu_g \nabla_X L_b(X(t), \lambda(t), \mu(t)) - \mu_t * (X(t) - T_X(t)) \tag{31}$$

$$\frac{d}{dt}\lambda(t) = \nabla_\lambda L_b(X(t), \lambda(t), \mu(t)) \tag{32}$$

$$\frac{d}{dt}\mu(t) = \nabla_\mu L_b(X(t), \lambda(t), \mu(t)) \tag{33}$$

where μ_g and μ_t are constants controlling the relative weights between local search and global exploration.

Note that our proposed method is a trajectory-based method, but differs from existing trajectory-based methods that rely on internal forces to modify the trajectory. Instead, it uses an *external* force (a traveling problem-independent trace) to pull the trajectory out of local minima.

The design of a good trace function $T_X(t)$ is very important because our method relies on it to travel through the solution space. Four criteria have been considered up to now. First, the trace should be aperiodic so that it does not return to the same starting points and regenerates possibly the same trajectory. Second, the trace needs to be continuous in order to be differentiable. This allows the generated trajectory to follow the terrain in a continuous manner without restarting from new starting points. Third, the trace should be bounded so that it will not explore unwanted regions. Last, the trace should be designed to travel from coarse to fine so that it examines the search space in greater details when more time is allowed.

Since an analytic approach to design a good trace function is intractable, we have studied some heuristic functions and fine-tuned them [25]. In the following, we summarize our observations.

- Our trace-based method is a fine-level global search in the sense that the search space covered by a trace grows linearly with the length of the trace (and, therefore, the time to complete the algorithm). However, a search space grows exponentially with respect to the number of dimensions. Hence, a trace-based method does not give good coverage when the search space is large. To overcome this limitation, the method may need to be combined with other coarse-level global searches, such as simulated annealing and genetic algorithms. This hybrid approach does not work well for constrained optimization problems because existing coarse-level global-search algorithms

generally have difficulty in satisfying constraints. Starting from points where constraints are violated usually do not lead to good feasible solutions.

- Our trace-based method relies on the distance between the current position of the trace and that of the trajectory to pull the trajectory out of local saddle points. When local gradients are very large, the external force due to the trace may not be enough to pull the trajectory out of local saddle points. To address this problem, we have developed a dynamic variable scaling method that scales a variable dimension when the gradient of the trajectory in this dimension exceeds a threshold (e.g. 10^2). Scaling not only reduces the local gradient of the trajectory but also increases the distance between the trace and the trajectory, thereby providing adequate force to pull the trajectory out of local saddle points. By controlling the scaling factor, we are able to explore a larger search space.

Based on substantial experiments, we have designed an aperiodic trace function as follows.

$$T_i(t) = \rho \sin\left[2\pi \left(\frac{t}{2}\right)^{0.95+0.45i-1/n} + 2\pi \frac{i-1}{n}\right] \tag{34}$$

where i represents the ith dimension, ρ is a coefficient specifying the range, and n is the dimension of the original variable space X.

We have described our trace-based method using one trace function. In general, the method can be cascaded, using the output trajectory of one stage as the trace of the next stage. This bootstrapping allows trajectories in later stages to go deeper into a local region, thereby providing better starting points for the local-search phase. In Figure 7, we have shown three stages of the global-search phase, each of which outputs a trajectory based on (31). In the first stage of the global search, the user-defined trace function $T_X(t)$ leads the trajectory to form Trajectory 1 in Figure 7. In the second and third stages of the global search, the trace function $T_X(t)$ is the trajectory from the previous stage. Using the trajectories output from the three stages, we identify a set of promising starting points and perform local Lagrangian searches from them. The final result is the best solution among all these local searches. In our implementation, when an output trajectory is a collection of discrete sample points of a continuous trajectory, interpolations are performed to form a continuous trace for the next stage.

Generally, weights μ_g and μ_t can have different values in different global stages. For example, μ_t can be set to have large values relative to μ_g in the first stage so that global search is emphasized. In later stages, μ_g can have larger values, and the search is more focused on a local region. In the simplest case, μ_g and μ_t are set to constants and have the same values in all the stages. In our experiments, we have set $\mu_g a = 1$ and $\mu_t = 20$ for all three stages.

We have implemented our *MaxQ*, adaptive weighting, and trace-based search in a prototype *Novel* (Nonlinear Optimization Via External Lead) that extends our previous work in trace-based search [26, 32]. The prototype can solve both nonlin-

ear constrained as well as unconstrained optimization problems. It has been applied to solve design problems in signal processing [33], neural-network learning [26], and benchmark problems in operations research. Results on the latter are presented in the next section.

5. Experimental results

In this section we describe our experimental results on some existing benchmarks [6]. These benchmarks are challenging because they model practical applications that have been studied extensively in the past. As a result, improvements are generally difficult.

We use a common set of parameters, including step size and trace function, to solve all the problems. The reason for not tuning the parameters is to avoid any bias, as good solutions can always be obtained by sufficient tuning. We further set the starting points of *Novel* as those suggested in the benchmark set. The only exception is the problem-specific search range of the trace, which we set manually based on the solutions reported in the benchmarks. In practice, this is reasonable as search ranges are generally known. In cases that the search range is not available, we use trial and error, starting from a small range and gradually increasing it until no improvement in solutions can be found.

In the global-search phase of *Novel*, we used three stages that produce three trajectories (see Figure 7). Using $\mu_g = \mu_t = 1.0$, we chose 100 starting points from each trajectory based on their Lagrangian values for Lagrangian searches. After 300 Lagrangian searches, we report the best solution.

Table 1 summarizes the results found by *Novel*. Column 1 lists the problem identifications that appear in the benchmark collection [6]. Column 2 shows the problem-dependent search range that the trace covers. Column 3 shows the best known solutions reported in [6], and Column 4, the solutions reported by Epperly [5]. Here, symbol '$-$' means that the method is unable to find a solution for the corresponding problem. Column 5 shows the results obtained by *Novel* using the slack-variable method where the dynamic weight-adaptation strategy described in Section 3 is used. Without using adaptive weights, more than half of these problems cannot be solved due to divergence and oscillations described in Section 2.1. The last column shows the results obtained by *Novel* using *MaxQ*. Results in bold font are improved by *Novel* over the best known results, with improvements of up to 10%. Our results indicate that *Novel* is robust in discovering new regions and in escaping from local traps.

6. Conclusions

In this paper, we have studied three strategies to improve Lagrangian searches for solving nonlinear constrained optimization problems. First, we have studied a new method called *MaxQ* to convert inequality constraints into equality con-

Table 1. Results on a collection of constrained optimization benchmarks [6] comparing *Novel* using *MaxQ*, *Novel* using the slack-variable method, and Epperly's method [5]. Search times are in CPU seconds on a Sun SS 10/51. Improved solutions found by *MaxQ* are indicated in bold font. Symbol '–' means that the method was not able to find a solution for the corresponding problem.

Problem ID	*Novel* Search Range	Best Known Solutions	Epperly's Solutions	Slack Variable Solutions	*MaxQ* Solutions
2.1	1.0	−17.00	−17.00	−17.00	−17.00
2.2	10.0	−213.00	−213.00	−213.00	−213.00
2.3	10.0	−15.00	−15.00	−15.00	−15.00
2.4	10.0	−11.00	−11.00	−11.00	−11.00
2.5	1.0	−268.00	−268.00	−268.00	−268.00
2.6	1.0	−39.00	−39.00	−39.00	−39.00
2.7(1)	40.0	−394.75	−394.75	−394.75	−394.75
2.7(2)	40.0	−884.75	−884.75	−884.75	−884.75
2.7(3)	40.0	−8695.00	−8695.00	−8695.00	−8695.00
2.7(4)	40.0	−754.75	−754.75	−754.75	−754.75
2.7(5)	40.0	−4150.40	−4150.40	−4150.40	−4150.40
2.8	25.0	15990.00	15990.00	**15639.00**	**15639.00**
3.1	5000.0	7049.25	–	7049.25	7049.25
3.2	50.0	−30665.50	−30665.50	−30665.50	−30665.50
3.3	10.0	−310.00	−310.00	−310.00	−310.00
3.4	5.0	−4.00	−4.00	−4.00	−4.00
4.3	5.0	−4.51	−4.51	−4.51	−4.51
4.4	5.0	−2.217	−2.217	−2.217	−2.217
4.5	5.0	−11.96	**−13.40**	**−13.40**	**−13.40**
4.6	5.0	−5.51	−5.51	−5.51	−5.51
4.7	5.0	−16.74	−16.74	−16.75	−16.75
5.2	50.0	1.567	–	1.567	1.567
5.4	50.0	1.86	–	1.86	1.86
6.2	100.0	400.00	400.00	400.00	400.00
6.3	100.0	600.00	600.00	600.00	600.00
6.4	100.0	750.00	750.00	750.00	750.00
7.2	100.0	56825.00	–	56825.00	56825.00
7.3	150.0	46266.00	–	46266.00	**44903.00**
7.4	150.0	35920.00	–	35920.00	35920.00

straints. A Lagrangian search using the new constraints approaches saddle points on boundaries of feasible regions without oscillations. This overcomes the problems of oscillations and divergence when inequality constraints are converted by adding slack variables. Second, we have developed a method to adaptively adjust the relative weights between the objective and the constraints in a Lagrangian formulation. We show that adaptive weighting can improve the convergence speed of Lagrangian methods, without affecting the solution quality. Finally, we have applied a trace-based search to bring a trajectory from one saddle point into another in a continuous fashion. Our method generates information-bearing trajectories in its global search based on a user-defined trace function, and samples these trajectories for good starting points in its local search. This overcomes the problem of using random restarts when the trajectory is already in the vicinity of good saddle points. We have applied *MaxQ*, adaptive weighting, and trace-based search in *Novel* [26, 32], a global optimization system we have developed earlier to solve constrained as well as unconstrained optimization problems.

We have tested many benchmark problems derived from manufacturing, computed aided design, and other engineering applications and have compared *MaxQ* to the method based on slack variables [14] and that of Epperly [5]. Our results show that *MaxQ* is more robust in convergence and has found solutions that are either better than or the same as existing solutions. Our future work in this area will be on finding better trace functions, parallelizing the execution on massively parallel computers, and studying other challenging applications in neural-network learning and signal processing.

Acknowledgments

Research supported by National Science Foundation Grants MIP 92-18715 and MIP 96-32316, and by the Computational Science and Engineering Program, University of Illinois, Urbana-Champaign.

References

1. Ben-Tal, A., Eiger, G. and Gershovitz, V. (1994), Global minimization by reducing the duality gap, *Mathematical Programming* 63: 193–212.
2. Bohachevsky, I.O., Johnson, M.E. and Stein, M.L. (1986), Generalized simulated annealing for function optimization, *Technometrics* 28: 209–217.
3. Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987), Minimizing multimodal functions of continuous variables with the simulated annealing algorithm, *ACM Trans. Math. Software* 13: 2–280.
4. Diener, I. and Schaback, R. (1990), An extended continuous Newton method, *Journal of Optimization Theory and Applications* 67(1): 57–77.
5. Epperly, T. (1995), *Global Optimization of Nonconvex Nonlinear Programs Using Parallel Branch and Bound*. Ph.D. Thesis, University of Wisconsin, Madison.
6. Floudas, C.A. and Pardalos, P.M. (1990), *A Collection of Test Problems for Constrained Global Optimization Algorithms*, vol. 455 of *Lecture Notes in Computer Science*. Springer Verlag.

7. Floudas, C.A. and Pardalos, P.M. eds. (1992), *Recent Advances in Global Optimization*. Princeton University Press.
8. Fogel, D.B. (1994), An introduction to simulated evolutionary optimization, *IEEE Trans. Neural Networks* 5(1): 3–14.
9. Hansen, E.R. (1992), *Global Optimization Using Interval Analysis*. Marcel Dekker, New York.
10. Horst, R. and Tuy, H. (1993), *Global Optimization: Deterministic Approaches*. Springer Verlag.
11. Ingber, L. (1995), *Adaptive Simulated Annealing (ASA)*. Lester Ingber Research.
12. Jones, A.E.W. and Forbes, G.W. (1995), An adaptive simulated annealing algorithm for global optimization over continuous variables, *Journal of Optimization Theory and Applications* 6: 1–37.
13. Lucidi, S. and Piccioni, M. (1989), Random tunneling by means of acceptance–rejection sampling for global optimization, *Journal of Optimization Theory and Applications* 62: 255–277.
14. Luenberger, D.G. (1984), *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company
15. Michalewicz, Z. (1994), *Genetic Algorithms + Data Structure = Evolution Programs*. Springer Verlag.
16. Mockus, J. (1989), *Bayesian Approach to Global Optimization*. Kluwer Academic Publishers, Dordrecht–Boston–London.
17. Mockus, J. (1994), Application of bayesian approach to numerical methods of global and stochastic optimization, *Journal of Global Optimization* 4: 347–365.
18. Pardalos, P.M. (1993), *Complexity in Numerical Optimization*. World Scientific, Singapore and River Edge, N.J.
19. Pardalos, P.M. and Rosen, J.B. (1987), *Constrained Global Optimization: Algorithms and Applications*, vol. 268 of *Lecture Notes in Computer Science*. Springer Verlag.
20. Patel, N.R., Smith, R.L. and Zabinsky, Z.B. (1988), Pure adaptive search in Monte Carlo optimization, *Mathematical Programming* 43: 317–328.
21. Piccioni, M. (1987), A combined multistart-annealing algorithm for continuous global optimization. Technical Report 87–45, Systems and Research Center, The University of Maryland, College Park, MD.
22. Romeijn, H.E. and Smith, R.L. (1994), Simulated annealing for constrained global optimization, *Journal of Global Optimization* 5(2): 101–126.
23. Sarma, M.S. (1990), On the convergence of the Baba and Dorea random optimization methods, *Journal of Optimization Theory and Applications* 66: 337–343.
24. Schoen, F. (1991), Stochastic techniques for global optimization: A survey on recent advances, *Journal of Global Optimization* 1(3): 207–228.
25. Shang, Y. (1997), *Global Search Methods for Solving Nonlinear Optimization Problems*. Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL.
26. Shang, Y. and Wah, B.W. (1996), Global optimization for neural network training, *IEEE Computer* 29: 45–54.
27. Stuckman, B.E. (1988), A global search method for optimizing nonlinear systems, *IEEE Trans. on Systems, Man, and Cybernetics* 18(6): 965–977.
28. Törn, A. and Viitanen, S. (1992), Topographical global optimization, in C.A. Floudas and P.M. Pardalos (eds.), *Recent Advances in Global Optimization*, pp. 385–398. Princeton University Press.
29. Törn, A. and Žilinskas, (1989) *Global Optimization*. Springer Verlag.
30. Vanderbilt, D. and Louie, S.G. (1984), A Monte Carlo simulated annealing approach to optimization over continuous variables, *Journal of Computational Physics* 56: 259–271.
31. Žilinskas, A. (1992), A review of statistical models for global optimization, *Journal of Global Optimization* 2: 145–153.

32. Wah, B.W. and Chang, Y.-J. (1997), Trace-based methods for solving nonlinear global optimization problems, *Journal of Global Optimization* 10(2): 107–141.
33. Wah, B.W., Shang, Y., Wang, T. and Yu, T. (1997), Global optimization of QMF filter-bank-design using NOVEL, in *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, vol. 3, pp. 2081–2084. IEEE, April 1997.
34. Wah, B.W., Wang, T., Shang, Y. and Wu, Z. (1997), Improving the performance of weighted Lagrange-multiple methods for constrained nonlinear optimization. In *Proc. 9th Int'l Conf. on Tools for Artificial Intelligence*, pages 224–231. IEEE, November 1997.
35. Zabinsky, Z.B. and Smith R.L. (1992), Pure adaptive search in global optimization, *Mathematical Programming* 53: 323–338.
36. Zabinsky, Z.B. et al. (1993), Improving hit-and-run for global optimization, *Journal of Global Optimization* 3: 171– 192.