Improving the Performance of Weighted Lagrange-Multiplier Methods for Nonlinear Constrained Optimization*

Benjamin W. Wah, Tao Wang, Yi Shang, and Zhe Wu

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
URL: http://manip.crhc.uiuc.edu

Abstract

Nonlinear constrained optimization problems can be solved by a Lagrange-multiplier method in a continuous space or by its extended discrete version in a discrete space. These methods rely on gradient descents in the objective space to find high-quality solutions, and gradient ascents in the Lagrangian space to satisfy the constraints. The balance between descents and ascents depends on the relative weights between the objective and the constraints that indirectly control the convergence speed and solution quality of the method. To improve convergence speed without degrading solution quality, we propose an algorithm to dynamically control these relative weights. Starting from an initial weight, the algorithm automatically adjusts the weights based on the behavior of the search progress. With this strategy, we are able to eliminate divergence, reduce oscillations, and speed up convergence. We show improved convergence behavior of our proposed algorithm on both nonlinear continuous and discrete problems.

1 Introduction

Many applications in engineering, decision science and operation research are formulated as optimization problems. The *constrained nonlinear optimization problems* studied here take the following form.

$$\begin{array}{ll} \textit{Minimize} & f(X) \\ \textit{subject to} & g(X) \leq 0 \quad X = (x_1, x_2, \ldots, x_n) \\ & h(X) = 0 \end{array} \qquad (1)$$

where X is a vector of real numbers in continuous problems or a vector of discrete numbers in discrete problems, $f(X)$ is an objective function, $g(X) = [g_1(X), \cdots, g_k(X)]^T$ is a set of k inequality constraints, and $h(X) = [h_1(X), \cdots, h_m(X)]^T$ is a set of m equality constraints. Note that f, g, and h can be either continuous or discrete functions.

The problem defined in (1) can be solved by a large number of existing approaches [2] that can be classified into local and global searches. Local-search algorithms include gradient descent, Newton's method, conjugate-gradient method, and Lagrange-multiplier method. Starting from some initial point, they stop at a local minimum. Since the local minima found by local-search methods may be much worse than the global minimum for nonlinear problems, global-search methods have been studied to perform both global search and local refinement. The global-search component goes through the possible search space and identifies some promising points for regions that may have good solutions, whereas the local-refinement component uses these promising points as starting points and employs a local-search algorithm to find local minima.

In this paper, we focus on the Lagrange-multiplier method as a local-search method to find satisfiable solutions and show how its convergence speed can be improved.

2 Lagrange-Multiplier Methods

Lagrange-multiplier methods introduce Lagrange multipliers to gradually resolve constraints iteratively. It is an exact method that optimizes the objective $f(X)$ to meet the Kuhn-Tucker conditions [2].

Since Lagrangian methods cannot directly deal with inequality constraints $g_i(X) \leq 0$ in general cases, we transform inequality constraints into equality constraints by adding slack variables z_i, which results in $p_i(X) = g_i(X) + z_i^2 = 0$. The corresponding *Lagrangian function* and *augmented Lagrangian function*

*Research supported by National Science Foundation Grant MIP 96-32316 and National Aeronautics and Space Administration Contract NAG 1-613.

Proc. IEEE International Conference on Tools with Artificial Intelligence, 1997.

are defined as follows.

$$\mathcal{L}(X,\lambda,\mu) = f(X) + \lambda^T h(X) + \mu^T p(X) \quad (2)$$

$$L(X,\lambda,\mu) = f(X) + \lambda^T h(X) + ||h(X)||_2^2 + \mu^T p(X) + ||p(X)||_2^2 \quad (3)$$

where $\lambda = [\lambda_1, \cdots, \lambda_m]^T$ and $\mu = [\mu_1, \cdots, \mu_k]^T$ are two sets of Lagrange multipliers, and $p(X) = [p_1(X), \cdots, p_k(X)]^T$. We use the augmented Lagrangian function in this paper since it provides better numerical stability. After simplification [2], the augmented Lagrangian function becomes

$$L_z(X,\lambda,\mu) = f(x) + \lambda^T h(X) + ||h(X)||_2^2 + \sum_{i=1}^{k} \left[max^2(0, \mu_i + g_i(X)) - \mu_i^2 \right] \quad (4)$$

2.1 Continuous Lagrangian Methods

For continuous problems, we assume that in (1), every variable x_i $(i = 1, 2, \cdots, n)$ takes a value from R, and that $f(X)$, $g(X)$ and $h(X)$, as well as their derivatives, are continuous functions.

According to classical optimization theory [2], the following set of equations can be used to find local and global extrema of (4):

$$\nabla_X L_z(X,\lambda,\mu) = 0 \qquad \nabla_\lambda L_z(X,\lambda,\mu) = 0$$
$$\nabla_\mu L_z(X,\lambda,\mu) = 0 \quad (5)$$

These conditions are necessary to guarantee the (local) optimality to the solution of (1) and (3). Because a local minimum of the Lagrangian function $L_z(X,\lambda,\mu)$ is a local minimum in the original-variable (X) space and a local maximum of $L_z(X,\lambda,\mu)$ in the Lagrange-multiplier (λ and μ) space, it can be obtained by solving the following dynamic system of equations:

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

which perform descents in the original-variable space of X and ascents in the Lagrange-multiplier space of λ and μ. The dynamic system evolves over time t, and reaches equilibrium where all gradients vanish.

2.2 Discrete Lagrangian Methods (DLM)

For discrete optimization problems, all the variables x_i $(i = 1, 2, \cdots, n)$ take discrete values (e.g., integers). Given Lagrange multipliers λ and μ, we define the change in the Lagrangian function $L_z(X,\lambda,\mu)$ in the discrete variable space to achieve the maximum decrease in $L_z(X,\lambda,\mu)$ as follows [3]:

$$\Delta_X L_z(X,\lambda,\mu) \;=\; \delta_X \quad (7)$$

where $L_z(X \ominus \delta_X, \lambda, \mu)$[1] is the minimum value among all $L_z(Y,\lambda,\mu)$ for Y sufficiently close to X.

Having defined $\Delta_X L_z(X,\lambda,\mu)$, we seek discrete saddle points.[2] The iterative equations are as follows.

$$X(k+1) = X(k) \ominus \Delta_X L_z(X(k),\lambda(k),\mu(k))$$
$$\lambda(k+1) = \lambda(k) + c_1 h(X(k)) \quad (8)$$
$$\mu_i(k+1) = \mu_i(k) \begin{cases} +\, c_2 g_i(X(k)) \\ \quad \text{if} \;\; \mu_i(k) + g_i(X(k)) > 0 \\ -\, c_2 \mu_i(k) \\ \quad \text{if} \;\; \mu_i(k) + g_i(X(k)) \le 0 \end{cases}$$

where c_1 and c_2 are positive real numbers controlling how fast the Lagrange multipliers change. For brevity, the proofs showing the correctness of DLM and (8) are omitted here [3].

3 Convergence Speed of Lagrangian Methods

We show in this section that the convergence speed and/or the solution quality can be affected by an additional weight w in the objective part of the Lagrangian function. This results in a new Lagrangian function as follows,

$$L_o(X,\lambda,\mu) = w\, f(x) + \lambda^T h(X) + ||h(X)||_2^2 + \sum_{i=1}^{k} \left[max^2(0, \mu_i + g_i(X)) - \mu_i^2 \right] \quad (9)$$

where $w > 0$ is a static weight on the objective. When $w = 1$, $L_o(X,\lambda,\mu) = L_z(X,\lambda,\mu)$, which is the original Lagrangian function.

In general, adding a weight to the objective changes the Lagrangian function, which in turn may cause the dynamic system to settle at a different local minimum with different solution quality. This is especially true when the local minimum is on the boundary of a feasible region. In this section we show that the solution quality and convergence time can be influenced greatly by the choice of the initial weight.

Starting from initial point $(X(0), \lambda(0), \mu(0))$, (6) is used to find local minima satisfying the constraints

[1] $\ominus$ is an operator for changing one point in discrete space into another with δ value. An example of $\ominus$ is the exclusive-OR operator.

[2] We look for saddle points in discrete space because the first-order conditions defined in (5) do not exist in discrete space.

where L_z is replaced by L_o.

$$\begin{aligned} \frac{d}{dt}X(t) &= -\nabla_X L_o(X(t),\lambda(t),\mu(t)) \\ \frac{d}{dt}\lambda(t) &= \nabla_\lambda L_o(X(t),\lambda(t),\mu(t)) \\ \frac{d}{dt}\mu(t) &= \nabla_\mu L_o(X(t),\lambda(t),\mu(t)) \end{aligned} \tag{10}$$

We solve this dynamic system using an ordinary differential equation solver *LSODE*[3] and observe a search trajectory $(X(t),\lambda(t),\mu(t))$. When a local minimum is on the boundary of the feasible region (which is true for most problems studied), the dynamic equation approaches it from both inside and outside of the feasible region. We observe four possible behaviors of the search trajectory.

- The trajectory converges without oscillations;
- 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.

Obviously, the first two cases are desirable, and the other two are not. Moreover, we would like to reduce the amount of oscillations and improve convergence time by proper control of w.

To illustrate the first behavior, consider the following simple example.

$$\begin{aligned} \textit{Minimize} \quad & x^2 \\ \textit{subject to} \quad & x \le -10 \end{aligned} \tag{11}$$

When we start from the initial point $X(t=0)=-20$ with $\mu(t=0)=0$ using $w=1$, we obtain a trajectory without oscillations, as shown in Figure 1.

To illustrate the last three behaviors (divergence, oscillations without convergence, and reduction of oscillations until convergence), consider Problem 2.3 in [1]. We set $X(t=0)$, the initial point at $t=0$, at the middle of the search space, and $\lambda(t=0)=\mu(t=0)=0$. The total time used for *LSODE* is $t_{max}=10^5$, which is divided into small units of $\Delta t = 1.0$, resulting in a maximum of 10^5 iterations ($= t_{max}/\triangle t$). The stopping condition for (10) is

$$||dX(t)/dt||^2 + ||d\lambda(t)/dt||^2 + ||d\mu(t)/dt||^2 \le \delta \tag{12}$$

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

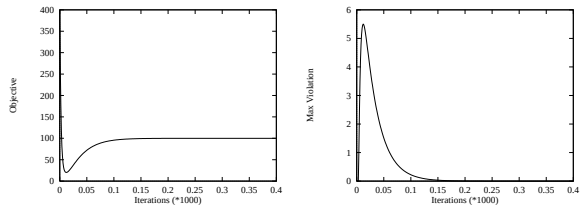


Figure 1: Objective function and maximum violation converge without oscillations.

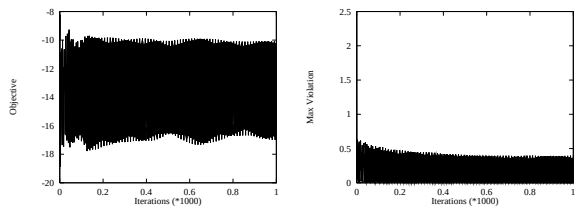


Figure 2: Objective function and maximum violation oscillate.

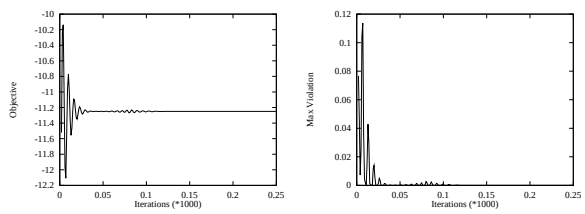


Figure 3: Objective function and maximum violation converge after oscillations subsided.

where $\delta = 10^{-25}$ in our experiments. The dynamic system stops when it converges or when it reaches the maximum number of iterations.

When $w=1$, (10) diverges quickly into infinity, meaning that the original Lagrangian method governed by (6) will diverge. If we scale the objective by 10 (*i.e.*, $w=1/10$), then $f(X(t))$ oscillates within the range -17 and -10, while the maximum violation $v_{max}(t)$ is between 0 to 0.4, as shown in Figure 2. Here, $v_{max}(t)$ at time t is defined as

$$v_{max}(t) = \max_{\substack{1\le i\le m,\\ 1\le j\le k}} \{|h_i(X(t))|, \max[0, g_j(X(t))]\} \tag{13}$$

If we further reduces w to $1/15$, then the oscillations are reduced, and the trajectory eventually converges (see Figure 3).

To understand the effect of w on convergence, we randomly generated 20 starting points $X(t=0)$ uniformly distributed in the search space of Problem 2.3 with all Lagrange multipliers $\lambda(t=0)=\mu(t=0)=0$. For each starting point, we tried different static weight w. In addition, for cases that converged, we report the average convergence time and the solution quality.

Table 1: Effect of w on convergence time and solution quality for Problem 2.3.1 in [1]. (The convergence behavior is measured by three integers: the number of diverged trajectories, the number of oscillating trajectories, and the number of converged trajectories.)

Weight w	Converge Behavior	Average Time(sec.)	Average Solution
1	20/0/0	-	-
1/3	20/0/0	-	-
1/5	0/20/0	-	-
1/7	0/20/0	-	-
1/9	0/20/0	-	-
1/11	0/10/10	-	-
1/15	0/0/20	2.88	-12.67
1/30	0/0/20	1.72	-12.67
1/60	0/0/20	1.76	-12.67
1/150	0/0/20	1.87	-12.67
1/500	0/0/20	1.98	-12.67
1/1000	0/0/20	2.20	-12.67
1/10000	0/0/20	3.90	-12.67
1/100000	0/0/20	2875.1	-12.67

Table 1 shows the results. When $w \leq \frac{1}{3}$, the trajectory diverges for every starting point. When w is between $\frac{1}{5}$ and $\frac{1}{9}$, the trajectories always oscillate without convergence. For $w = \frac{1}{11}$, half of the trajectories finally converge, and the remaining oscillate. When $w \leq \frac{1}{15}$, all the trajectories converge. For cases that converge, the number of oscillations gradually reduces to zero as w is reduced. Hence, it is possible to eliminate all the oscillations at the expense of longer convergence time (e.g. 2875.1 seconds). The convergence behavior when $w = \frac{1}{30}$ seems to be good because both the number of oscillations and the convergence time are small.

The examples here illustrate that the convergence behavior can be affected dramatically by the choice of w. Moreover, the best choice is problem-instance dependent and cannot be selected a priori. Hence, we will propose a scheme to adapt w dynamically so that the convergence behavior is robust and predictable, irrespective of the choice of the static w.

4 Dynamic Weight Adaptation

In this section, we propose a strategy to adapt weight w based on the behavior of the dynamic system (10) (both continuous and discrete versions) in order to obtain high-quality solutions with short convergence time.

4.1 General Weight-Adaptation Strategy

Figure 4 outlines the algorithm. Its basic idea is to first estimate the initial weight $w(t = 0)$ (Step 1),

1. Set control parameters:
   time interval Δt
   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** $j \leq i_{max}$ **and** stopping condition is unsatisfied **do**
5. advance trajectory by Δt 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 data for calculating performance metrics
8. **if** $((j \ mod \ N_w) == 0)$ **then**
9. compute performance metrics
10. change w based on rules in Figure 5
    end if
11. **end while**

Figure 4: Framework of a new dynamic weight-adaptation algorithm

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 Δt so that the maximum number of iterations is $t_{max}/\Delta t$. 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 (13).

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 mth window $(m = 1, 2, \cdots)$, we calculate the average (or median) of $v_{max,i}$ over all the iterations in the window,

$$\bar{v}_m = \frac{1}{N_w} \sum_{j=(m-1)N_w+1}^{mN_w} v_{max,j} \tag{14}$$

and the average (or median) of the objective $f_i(X)$.

$$\bar{f}_m = \frac{1}{N_w} \sum_{j=(m-1)N_w+1}^{mN_w} f_j(X) \tag{15}$$

During the search, we apply an algorithm to solve the dynamic system (10), and advance the trajectory by time interval Δt 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

$v_{max,j}$ is larger than an extremely large 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 $v_{max,j}$ and $f_j(X)$, that 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 (14) and (15) (Step 9). Currently, we use the averages (or medians) of maximum violation $v_{max,i}$ and objective $f_j(X)$. In general, other application-specific metrics can be used, such as the number of oscillations of the trajectory in nonlinear continuous problems. Based on these measurements, we adjust w accordingly (Step 10).

In the next subsection, we discuss the specific weight-adaptation algorithm in Step 10 for continuous and discrete problems.

4.2 Dynamic Weight Adaptation for Continuous and Discrete Problems

To understand how weights should be updated in Step 10, we examine all possible behaviors of the resulting search trajectory in successive windows. We have identified four possible cases.

First, the trajectory does not stay within a feasible region, but goes from one feasible region to another through an infeasible region. During this time, the maximum violation $v_{max,i}$ is zero when the trajectory is in the first feasible region, increased when it travels from the feasible region to an infeasible region, and decreased when going from the infeasible region to the second feasible region. No oscillations will be observed because oscillations normally occur around a local minimum in one feasible region, as explained in Section 3. In this case, no change of w is required.

Second, the search trajectory oscillates around a local minimum of a feasible region. This can be detected when the number of oscillations in each window is larger than some threshold. Figures 2 and 3 show two typical types of oscillations. To determine whether the oscillations will subside eventually, we compute $\bar{v}_m - \bar{v}_{m+1}$, the difference of the average values of the maximum violation $v_{max,i}$, for two successive windows m and $m+1$. If the difference is not reduced reasonably, then we assume that the trajectory does not converge, and w is then updated.

Third, the search trajectory moves very slowly within a feasible region. This happens when w is very small, and the constraints dominate the search process. As a result, the objective value is improved very slowly and may eventually converge to a poor value. This situation can be identified when the trajectory remains within a feasible region in two successive windows, and there is little improvement in the objective. Obviously, we need to increase w in order to speed up the improvement of the objective. If the objective remains unchanged, then the trajectory has converged, and no further modification of w is necessary.

Given performance measurements for the mth window
average (or median) of the maximum violation: $\bar{v}_m$
average (or median) of the objective: $\bar{f}_m$
number of oscillations: NO_m
and application-specific constants $\alpha_0, \alpha_1, \beta_0, \beta_1, \delta, \epsilon$
1. Increase weight $w \Longleftarrow w/\alpha_0$ **if**
$(c_{1,1})$ $\bar{v}_{m-1}, \bar{v}_m < \delta$
$(c_{1,2})$ $\beta_0 |\overline{f}_{m-1}| > \overline{f}_{m-1} - \overline{f}_m > \beta_1 |\overline{f}_{m-1}|$
2. Decrease weight $w \Longleftarrow \alpha_1 w$ **if**
$(c_{2,1})$ $\bar{v}_m \geq \delta$
$(c_{2,2})$ $\bar{v}_{m-1} - \bar{v}_m \leq \beta_0 \bar{v}_{m-1}$
$(c_{2,3})$ $NO_m \geq \epsilon$

Figure 5: Weight-adaptation rules for continuous and discrete problems (Step 10 of Figure 4).

Finally, the trajectory does not oscillate when it is started from a point in a feasible region, but rather goes outside the feasible region and then converges to a point on the boundary of the feasible region (see Figure 1 for example). In this case, a large w on the objective makes it more difficult to satisfy the constraints, causing the trajectory to move slowly back to the feasible region. At this time, it is desirable to reduce w to accelerate convergence. This situation happens frequently in some cases, such as QMF filterbank design problem [5], and appropriate decrease of w will greatly shorten the convergence time.

Given the four convergence behaviors, Figure 5 shows a comprehensive weight-adaptation algorithm for both continuous and discrete problems. Scaling factors $0 < \alpha_0, \alpha_1 < 1$ represent how fast w is updated. Because we use numerical methods to solve the dynamic system defined in (10), a trajectory in window m is said to satisfy all the constraints when $v_m < \delta$, where δ is related to the convergence condition and the required precision. Parameters $0 < \beta_0, \beta_1 < 1$ control, respectively, the degrees of improvement over the objective and the reduction of the maximum violation. Note that when comparing values between two successive windows $m-1$ and m, both must use the same weight w; otherwise, the comparison is not meaningful because the terrain may be totally different. Hence, after adapting w, we should wait at least two windows before changing it again.

Weight w should be increased (Rule 1) when we observe the third convergence behavior. In this case,

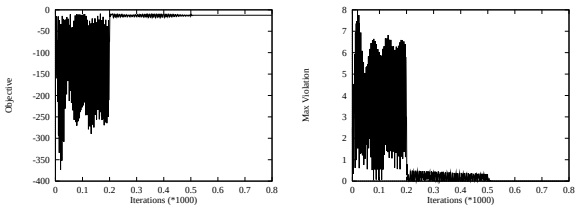


Figure 6: Objective function and maximum violation first oscillate and then converge using dynamic-weight adaptation.

the trajectory is within a feasible region, and the objective is improved in successive windows. Weight w will be increased when the objective does not improve fast enough in a feasible region, until an upper-bound improvement is reached.

Weight w should be decreased (Rule 2) when we observe the second convergence behavior (the trajectory oscillating around a local minimum) or the fourth convergence behavior (the trajectory moving slowly back to the feasible region). Weight w will be decreased when the trajectory oscillates, and the trend of the maximum violation does not decrease.

5 Illustrations of the Dynamic Weight-Adaptation Strategy

In this section, we illustrate our weight-adaptation algorithm using nonlinear continuous optimization problems, which include a set of benchmark problems [1, 4]. After some experimentation, we pick the time unit $\triangle t = 1$ in $LSODE$, and set the window size to be $N_w = 100$.

Recall in Figure 2 that the trajectory oscillates when $w = \frac{1}{10}$ for Problem 2.3 in [1]. Figure 6 shows the resulting trajectory and the maximum violation when the dynamic weight-adaptation algorithm is applied on the same problem. We started with $w(t=0) = \frac{1}{5}$, $\alpha_0 = \alpha_1 = \frac{1}{2}$, $\delta = 10^{-8}$, $\beta_0 = 10^{-2}$, and $\beta_1 = 10^{-3}$. In the first window (first $N_w = 100$ iterations), the average $\bar{v}_1 = 4.11$, which increases slightly to $\bar{v}_2 = 4.2$ in the second window. In addition, $\bar{v}_2 \geq \delta$ and $NO_2 \geq 5$. According to Rule 2 in Figure 5, w is updated to $\frac{1}{10}$. This change in w leads to significant reduction in the maximum violation in the third window. Weight w remains unchanged between the third and the fifth windows. At the end of the fifth window, the maximum violation changes very little. Hence, the algorithm reduces w to $\frac{1}{20}$, making the trajectory converges quickly.

To illustrate the improvement due to dynamic-weight adaptation, we apply the algorithm on the same problem studied in Table 1 from 20 randomly generated starting points. Table 2 shows the results, indicating the convergence of the search trajectory for each initial weight w. For cases that converge using static weights, dynamic-weight adaptation can reduce the convergence time, especially when the choice of the initial weight is poor. For instance, when the initial weight is $\frac{1}{100000}$, the search converges with the final weight in the range $[\frac{1}{6250}, \frac{1}{781.25}]$ and an average convergence time of 3.85 sec., much less than that the 2875.1 sec. required before. If the initial weight chosen is good (e.g. $w = \frac{1}{60}$), then the algorithm is able to maintain the original weight during the search and gets good convergence behavior. This shows the stability and robustness of our dynamic weight-adaptation algorithm, irrespective of the choice of the initial weight.

Table 2: Effects of weight adaptation on convergence time and solution quality for Problem 2.3.1 in [1]

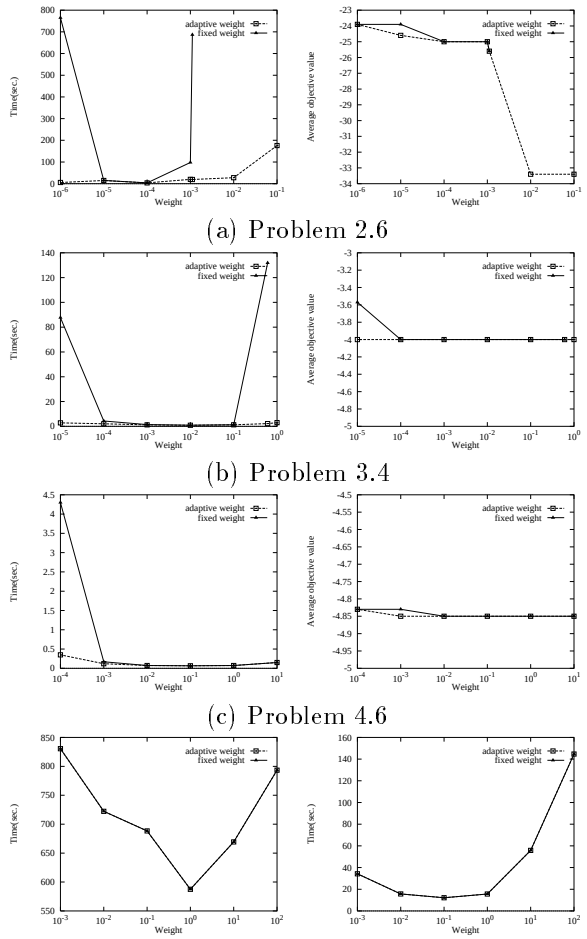
Starting Weight w	Final Weight Range $[w_l, w_u]$	Avg. Converge Time (sec.)	Average Solution
1	1/20	19.14	−13.55
1/3	1/15	10.70	−12.67
1/5	1/20	43.36	−12.67
1/7	[1/28,1/14]	27.43	−12.67
1/9	1/18	19.97	−12.67
1/11	1/22	4.22	−12.67
1/15	1/15	2.88	−12.67
1/30	1/30	1.72	−12.67
1/60	1/60	1.76	−12.67
1/150	1/150	1.87	−12.67
1/500	[1/500,1/1000]	1.95	−12.67
1/1000	[1/1000,1/500]	2.15	−12.67
1/10000	[1/5000,1/625]	3.34	−12.67
1/100000	[1/6250,1/781.25]	3.85	−12.67

6 Experimental Results

In this section we show further experimental results in applying our dynamic weight-adaptation algorithm to both nonlinear continuous and discrete optimization problems. We also compare our results to those obtained by Lagrangian methods with static weights.

6.1 Nonlinear Continuous Benchmark Problems

These constrained benchmark problems [1] were derived from a variety of engineering applications. We selected five problems (with identifiers 2.6, 3.4, 4.6, 5.4 and 6.4) from different classes to test our algorithm. In each problem, we randomly generated 20 starting points $X(t=0)$ uniformly distributed in its search range, and set the initial values for the Lagrange multipliers to be zero ($\lambda(t=0) = \mu(t=0) = 0$). For



(a) Problem 2.6

(b) Problem 3.4

(c) Problem 4.6

(d) Problem 5.4 (e) Problem 6.4

Figure 7: Comparison of the Lagrangian method with static weights and the adaptive Lagrangian method in terms of convergence time and solution quality.

each starting point, we applied the Lagrangian method using both static weights and dynamically changed weights under the same convergence condition defined in (12). Hence, the Lagrangian method stopped when it either reached the maximum iteration count i_{max} or satisfied the convergence condition.

We chose the static weights in the range between 10^{-6} to 10^{2}, which were also used as initial weights $w(t=0)$ in the dynamic algorithm. The weights were chosen from a wide range in order to test the robustness of our weight-adaptation algorithm. We would like our algorithm to adjust the weight so that the search will converge faster with a solution that is at least as good as that with static weights. In our experiments, we used the following control parameters: time unit $\Delta t = 1$, window size $N_w = 100$, $\alpha_0 = \alpha_1 = \frac{1}{2}$, $\delta = 10^{-8}$, $\beta_0 = 10^{-2}$, and $\beta_1 = 10^{-3}$.

We used two performance metrics, convergence time and solution quality, in our experiments, where convergence time was measured by the average convergence time (in seconds) from 20 starting points, and solution quality was measured by the average objective value when the search converged.

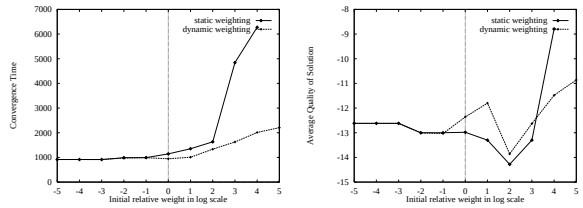
From the experimental results shown in Figure 7, we have the following observations. First, different problems require different ranges of static weights in order to get fast convergence. For example, Problem 2.6 (Figure 7a) converges the fastest using $w = 10^{-4}$, whereas Problem 5.4 (Figure 7d) converges the fastest using $w = 1$. These weights differ by four orders of magnitude. Hence, it is difficult to choose a good static weight for a given problem instance in advance.

Second, our dynamic weight-adaptation algorithm outperforms the Lagrangian method with static weights. For example, in Problem 2.6 (Figure 7a), the Lagrangian method with a static weight is unable to converge (either diverge or oscillate forever) when $w = 10^{-2}$ or larger. However, our dynamic weight-adaptation algorithm can detect this misbehavior and adjust w to allow the search to converge in 27.62 seconds on the average. Likewise, the Lagrangian method with static weight $w = 10^{-6}$ converges using an average of 765.0 seconds, but the dynamic algorithm converges using an average of 5.64 seconds and with the same solution quality. These results show that our weight-adaptation algorithm is robust and insensitive to the initial weights $w(t=0)$ in comparison with the Lagrangian method with a static weight. In fact, our dynamic algorithm can get even better solutions in shorter time when the initial weight is small.

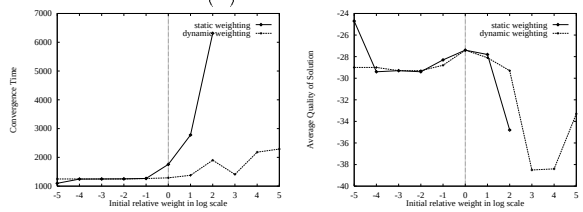
Third, when all the constraints are equality constraints, w is unchanged in the dynamic algorithm, resulting in the same performance for both methods. This is shown in Problem 5.4 (Figure 7d). In some cases, w is unchanged in the dynamic algorithm when there are both inequality and equality constraints, such as Problem 6.4 (Figure 7e). This happens when the Lagrange multipliers already take care of the balance between the objective and the constraints.

6.2 Nonlinear Integer Programming

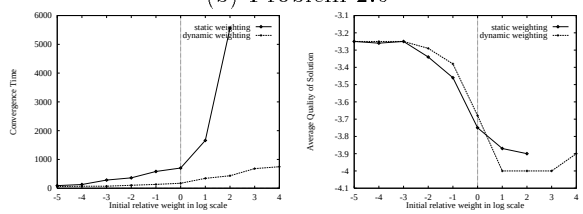
To test the dynamic weight-adaptation strategy, we selected three problems 2.3, 2.6 and 3.4 from the benchmark collection [1]. We first transformed them to discrete integer programming problems, and then randomly generated 20 discrete starting points with initial values of Lagrange Multipliers to be zero. For each starting point, we applied the Lagrangian method using both static weights and dynamically changed



(a) Problem 2.3



(b) Problem 2.6



(c) Problem 3.4

Figure 8: Comparison of the Lagrangian method with static weights and the adaptive Lagrangian method in terms of convergence time and solution quality for discrete benchmark problems

weights. We stopped the search when the convergence condition was satisfied or the maximal number of iterations was reached.

The initial weight w was in the range of $[10^{-5}, 10^{5}]$, which was large enough to test the robustness of our adaptive Lagrangian method. The parameters for our algorithm were: window size $N_w = 320, \alpha_0 = 0.8, \alpha_1 = 0.5, \delta = 10^{-8}, \beta_0 = 10^{-3}$, and $\beta_1 = 10^{-4}$.

As is in continuous problems, we use both convergence time and solution quality as our performance metrics, where convergence time is measured by the average convergence time from 20 starting points, and solution quality is measured by the average objective value when the search converges. Figure 8 shows the comparative results. Our dynamic weight-adaptation algorithm improves a lot in convergence time over the algorithm with static weights. When weight w is larger than 100.0 for Problem 2.6, the Lagrangian method with static weights was unable to converge even after 10 hours. In contrast, our dynamic weight-adaptation algorithm converged in less than 1 hour for all different initial weights.

7 Conclusions

In this paper we have studied and evaluated an efficient weight-adaptation algorithm for Lagrangian methods, which adjusts the relative weights between the objective and the constraints based on statistics collected during the search. We have applied the adaptive method to both continuous and discrete constrained optimization problems. (Due to space limitations, we are not able to show results on designing QMF filter banks and multiplierless QMF filter banks.) In both cases, the adaptive method always converges in much shorter time with either the same or better solution. Our results show that the adaptive method is robust, works for both continuous and discrete problems, and exhibits consistent and good convergence behavior.

References

[1] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[2] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1984.

[3] Y. Shang and B. W. Wah. A discrete lagrangian-based global-search method for solving satisfiability problems. *J. of Global Optimization*, (accepted to appear) 1997.

[4] B. W. Wah and Y.-J. Chang. Trace-based methods for solving nonlinear global optimization problems. *J. of Global Optimization*, 10(2):107–141, March 1997.

[5] B. W. Wah, Y. Shang, T. Wang, and T. Yu. Global optimization of QMF filter-bank design using NOVEL. In *Proc. Int'l Conf. on Acoustics, Speech and Signal Processing*, volume 3, pages 2081–2084. IEEE, April 1997.