Comput Optim Appl (2010) 47: 257–288
DOI 10.1007/s10589-008-9218-1

An algorithm for nonlinear optimization problems with binary variables

Walter Murray · Kien-Ming Ng

Received: 6 April 2007 / Revised: 22 October 2008 / Published online: 2 December 2008

Abstract One of the challenging optimization problems is determining the minimizer of a nonlinear programming problem that has binary variables. A vexing difficulty is the rate the work to solve such problems increases as the number of discrete variables increases. Any such problem with bounded discrete variables, especially binary variables, may be transformed to that of finding a *global* optimum of a problem in continuous variables. However, the transformed problems usually have astronomically large numbers of local minimizers, making them harder to solve than typical global optimization problems. Despite this apparent disadvantage, we show that the approach is not futile if we use smoothing techniques. The method we advocate first convexifies the problem and then solves a sequence of subproblems, whose solutions form a trajectory that leads to the solution. To illustrate how well the algorithm performs we show the computational results of applying it to problems taken from the literature and new test problems with known optimal solutions.

Keywords Nonlinear integer programming · Discrete variables · Smoothing methods

The research of W. Murray was supported by Office of Naval Research Grant N00014-08-1-0191 and Army Grant W911NF-07-2-0027-1.

W. Murray (✉)
Systems Optimization Laboratory, Department of Management Science and Engineering, Stanford University, Stanford, CA 94305-4022, USA
e-mail: walter@stanford.edu

K.-M. Ng
Department of Industrial & Systems Engineering, National University of Singapore, Singapore, Singapore
e-mail: isenkm@nus.edu.sg

1 Introduction

Nonlinear optimization problems whose variables can only take on integer quantities or discrete values abound in the real world, yet there are few successful solution methods and even one of the simplest cases with quadratic objective function and linear constraints is NP-hard (see, e.g., [6]). We focus our interest on problems with linear equality constraints and binary variables. However, once the proposed algorithm has been described it will be seen that extending the algorithm to more general cases, such as problems with linear inequality constraints, or problems with some continuous variables is trivial. Also, the binary variable requirement is not restrictive since any problem with bounded discrete variables can easily be transformed to a problem with binary variables.

The problem of interest can thus be stated as

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & Ax = b, \\ & x \in \{0, 1\}^n, \end{array} \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a twice continuously differentiable function, A is an $m \times n$ matrix with rank m and $b \in \mathbb{R}^m$ is a column vector. Assuming differentiability may seem strange when x is discrete. (Often $f(x)$ cannot be evaluated unless every $x_j = 0$ or 1.) Nonetheless for a broad class of real problems $f(x)$ is smooth and when it is not it is often possible to alter the original function suitably (see [26]). As noted problems in the form of (1.1) are generally NP-hard. There is no known algorithm that can be assured of obtaining the optimal solution to NP-hard problems for large problems within a reasonable amount of computational effort and time. It may even be difficult just to obtain a feasible solution to such problems. The challenge is to discover algorithms that can generate a good approximate solution to this class of problems for an effort that increases only slowly as a function of n.

If the requirement that $x \in \{0, 1\}^n$ is dropped then typically the effort to find a *local* minimizer of (1.1) increases only slowly with the size of the problem. It would seem attractive if (1.1) could be replaced by a problem in continuous variables. The equivalence of (1.1) and that of finding the *global* minimizer of a problem with continuous variables is well known (see [14, 34]). One simple way of enforcing $x \in \{0, 1\}^n$ is to add the constraints $0 \le x \le e$ and $x^{\mathrm{T}}(e - x) = 0$, where e is the vector of unit elements. Clearly for x_i to be feasible then $x_i = 0$ or 1. The difficulty is that there is a world of difference between finding a global minimizer as opposed to a local minimizer. If there are only a few local minimizers then the task is relatively easy but in this case it is possible that every vertex of the feasible region (and there could be 2^n vertices) is a local minimizer. Typically the minimizer found by an algorithm to find a local minimizer is entirely determined by the starting point chosen. Traditional methods for global optimization (see [25]) perform particularly poorly on problems with numerous local minimizers.

A number of approaches have been developed to handle mixed-integer nonlinear programming problems via a transformation of the discrete problem into a continuous problem. These approaches involve geometric, analytic and algebraic techniques,

including the use of global or concave optimization formulations, semidefinite programming and spectral theory (see e.g., [9, 19, 20, 30, 31]). In particular, [23, 29] give an overview of many of these continuous approaches and interior-point methods. Perhaps the most obvious approach is to simply ignore the discrete requirement, solve the resulting "relaxed" problem and then discretize the solution obtained by using an intelligent rounding scheme (see [16], Chapter 7 and [22]). Such approaches work best when the discrete variables are in some sense artificial. For example, when optimizing a pipe layout it is known that pipes are available only in certain sizes. In principle pipes could be of any size and the physics of the model is valid for sizes other than those available. However, even in these circumstances there may be difficult issues in rounding since it may not be easy to retain feasibility. It is not difficult to see that this approach can fail badly. Consider minimizing a strictly concave function subject to the variables being binary. The relaxed solution has a discrete solution and obviously there is no need to round. Rather than being good news it illustrates the solution found is entirely determined by the choice of the initial point used in the continuous optimization algorithm. Given that such methods find local minimizers the probability of determining the global minimizer is extremely small since there may be 2^n local minimizers. There are other problems in which little information can be determined from the continuous solution. For example, in determining the optimum location of substations in an electrical grid relaxing the discrete requirements results in a substation being placed at every node that precisely matches the load. In this case this is the global minimizer of the continuous problem but it gives almost no information on which node to place a substation. To be feasible it is necessary to round up to avoid violating voltage constraints. Moreover, for many nodes the loads are identical so any scheme that rounds up some variables and rounds down others in the hope of the solution still being feasible has trouble identifying a good choice.

There are even fewer methods that are able to solve general problems with a large number of variables. Three commercial packages that are available are DICOPT (see [18]), SBB (see [5]) and BARON (see [32]). DICOPT is a package for solving mixed-integer nonlinear programming problems based on extensions of the outer-approximation algorithm (see [10]) for the equality relaxation strategy and then solving a sequence of nonlinear programming and mixed-integer linear programming problems. SBB is based on a combination of the standard branch-and-bound method for the mixed-integer linear programming problems and standard nonlinear programming solvers. BARON is a global optimization package based on the branch-and-reduce method (see [33]). These solver packages can in principle solve large nonlinear mixed-integer programming problems. However, it will be seen that the performance of these algorithms, especially on large problems, are sometimes poor and there is a clear need for better algorithms for this class of problems. It may be that a single algorithm that can find good solutions for most large problems is a step too far, but it helps to add to the arsenal of solution methods new methods whose behavior and character differ from current methodology.

Throughout this paper, we let $\|\cdot\|$ denote the 2-norm, i.e., the Euclidean norm, of a vector, and let e denote the column vector of ones with dimension relevant to the context. If x and y are vectors in $\mathbb{R}^n$, we let $x \leq y$, $x < y$ to mean respectively that $x_i \leq y_i$, $x_i < y_i$ for every $i \in \{1, 2, \ldots, n\}$, and $\mathrm{Diag}(x)$ to denote the diagonal matrix

with the diagonal elements made up of the respective elements x_i, $i \in \{1, 2, \ldots, n\}$. Also, the null space of a matrix A is defined as $\{x \in \mathbb{R}^n : Ax = 0\}$. For a real-valued function f, we say that f is a C^k function if it is kth-order continuously differentiable.

2 An exact penalty function

Rather than add nonlinear constraints we have chosen to add a penalty term

$$\sum_{j \in J} x_j(1 - x_j), \tag{2.1}$$

with a penalty parameter $\gamma > 0$, where J is the index set of the variables that are judged to require forcing to a bound. We could have added the penalty $x^{\mathrm{T}}(e - x)$, but often the objective is nearly concave and if there are not many equality constraints some of the variables will be forced to be 0 or 1 without any penalty term being required. The problem then becomes

$$\begin{array}{ll} \text{Minimize} & F(x) \triangleq f(x) + \gamma \displaystyle\sum_{j \in J} x_j(1 - x_j) \\ \text{subject to} & Ax = b, \\ & 0 \le x \le e. \end{array} \tag{2.2}$$

In general, it is possible to show that under suitable assumptions, the penalty function introduced this way is "exact" in the sense that the following two problems have the same global minimizers for a sufficiently large value of the penalty parameter γ:

$$\begin{array}{ll} \text{Minimize} & g(x) \\ \text{subject to} & Ax = b, \\ & x \in \{0, 1\}^n \end{array} \tag{2.3}$$

and

$$\begin{array}{ll} \text{Minimize} & g(x) + \gamma x^{\mathrm{T}}(e - x) \\ \text{subject to} & Ax = b, \\ & 0 \le x \le e. \end{array} \tag{2.4}$$

An example of such a result (see for example [30]) is:

Theorem 2.1 *Let $g : [0, 1]^n \to \mathbb{R}$ be a C^1 function and consider the two problems (2.3) and (2.4) with the feasible region of (2.3) being non-empty. Then there exists $M > 0$ such that for all $\gamma > M$, problems (2.3) and (2.4) have the same global minimizers.*

Note that although this is an exact penalty function it is twice continuously differentiable and the extreme ill-conditioning arising out of the need for $\gamma \to \infty$ normally required for smooth penalty functions is avoided. Indeed, a modestly large value of γ is usually sufficient to indicate whether a variable is converging to 0 or 1.

Since the penalty term is concave we are highly likely to introduce local minimizers at almost all the feasible integer points, and just as significantly, saddle points at interior points. It is clearly critical that any method used to solve the transformed problem be one that can be assured of not converging to a saddle point.

The idea of using penalty methods for discrete optimization problems is not new (see, e.g., [3]). However, it is not sufficient to introduce only the penalty terms and hope to obtain the global minimizer by solving the resulting problem, because it is highly likely that many undesired stationary points and local minimizers are being introduced in the process. This flaw also applies to the process of transforming a discrete optimization problem into a global optimization problem simply by replacing the discrete requirements of the variables with a nonlinear constraint. The danger of using the penalty function alone is illustrated by the following example:

Example Consider the problem $\min\{x^2 : x \in \{0, 1\}\}$. It is clear that the global minimizer is given by $x^* = 0$. Suppose the problem has been transformed to $\min_{0\le x\le 1}\{x^2 + \gamma x(1 - x)\}$, where $\gamma > 0$ is the penalty parameter. If $\gamma > 2$ then the first-order KKT conditions are satisfied at 0, 1 and $\frac{\gamma}{2(\gamma-1)}$. Suppose in a descent method the initial point is chosen in the interval $[\frac{\gamma}{2(\gamma-1)}, 1]$, then the sequence generated will not converge to 0. Thus for large values of γ the probability of converging to 0 from a random initial point is close to 0.5. A probability of 0.5 may not seem so bad but if we now consider the n-dimensional problem $\min\{\|x\|^2 : x \in \{0, 1\}^n\}$, then every vertex is a local minimizer of the continuous problem and the probability of converging from a random initial point to the correct local minimizer is not much better than $1/2^n$.

In the following we assume that the feasible space is bounded, which is the case for the problems of interest.

Definition Let $\bar{x}$ be a minimizer of (2.2) and $S(\bar{x})$ denote the set of feasible points for which there exists an arc emanating from $\bar{x}$ such that the tangent to the arc at x is $Z^{\mathrm{T}}\nabla_x f(x)$, where the columns of Z are a basis for the null space of the constraints active at x. Let the volume of $S(\bar{x})$ be denoted by $S_v(\bar{x})$. We may rank the minimizers in terms of the size of $S_v(\bar{x})$.

If $x^* \in \mathbb{R}^n$ is the global minimizer and V is the volume of the feasible space then we can expect that

$$\lim_{n\to\infty} S_v(x^*)/V = 0.$$

It is this fact that makes it unlikely that simply applying a local search method to (2.2) will be successful. Indeed there is little to hope that a good local minimizer would be found. If the minima are uniformly distributed with mean M then the expected value of $f(\bar{x})$ for a random starting point is M. There is little reason to suppose M is close to $f(x^*)$.

There are alternative penalty functions. Moreover, $x_i(1 - x_i) = 0$ is a complementarity condition and there is a vast literature on how to impose these conditions. In addition to penalty functions there are other means of trying to enforce variables to

take their discrete values. A particularly useful approach is to replace terms such as $c^{\mathrm{T}}x$ that often appear in the objective, where $c_i > 0$ ($x_i = 1$ implies a capital cost), by the expression $\sum_{i=1}^{n} c_i(1 - e^{-\beta x_i})$ with $\beta = 10$ typically. This term favors setting variables to 0 and 1 rather than 0.5 and 0.5. It may be thought that it would always prefer all variables to be 0, but that is typically prevented by constraints. Also rounding up ensures feasibility but rounding down does not.

3 Smoothing methods

In general, the presence of multiple local minima in an optimization problem is common and often makes the search for the global minimum very difficult. The fewer the local minima, the more likely it is that an algorithm that finds a local minimizer will find the global minimum. *Smoothing methods* refer to a class of methods that replace the original problem by either a single or a sequence of problems whose objective is "smoother". In this context "smoother" may be interpreted as a function whose second or higher order derivatives are smaller in magnitude, with a straight line being the ultimate smooth function.

The concept of smoothing has already been exploited in nonconvex optimization and discussed in the context of global optimization (see, e.g., [19]). There are a number of ways to smooth a function and which is best depends to a degree on what characteristics of the original function we are trying to suppress. Smoothing methods can be categorized into two types: *local* or *global* smoothing. Local smoothing algorithms are particularly suited to problems in which noise arises during the evaluation of a function. Unfortunately, noise may produce many local minimizers, or it may produce a minimizer along the search direction in a linesearch method and result in a tiny step. Local smoothing has been suggested to eliminate poor minimizers that are part of the true problem. Under such circumstances the introduction of smoothing alters the problem and may change the required global minimizer. To overcome this, a sequence of problems is solved in which the degree of smoothing is reduced to zero. While local smoothing may be useful in eliminating tiny local minimizers even when there are large numbers of them, they are less useful at removing significant but poor minimizers (see [25]).

We would like to transform the original problem with many local minima into one that has fewer local minima or even just one local minimum and so obtaining a global optimization problem that is easier to solve (see Fig. 1). Obviously we wish to eliminate poor local minima. In the subsequent discussion, we consider modified objective functions of the form

$$F(x, \mu) = f(x) + \mu g(x),$$

where f and g are real-valued C^2 functions on $\mathbb{R}^n$ and $\mu \geq 0$.

3.1 Global smoothing

The basic idea of global smoothing is to add a strictly convex function to the original objective, i.e.,

$$\mathcal{F}(x, \mu) = f(x) + \mu \Phi(x),$$

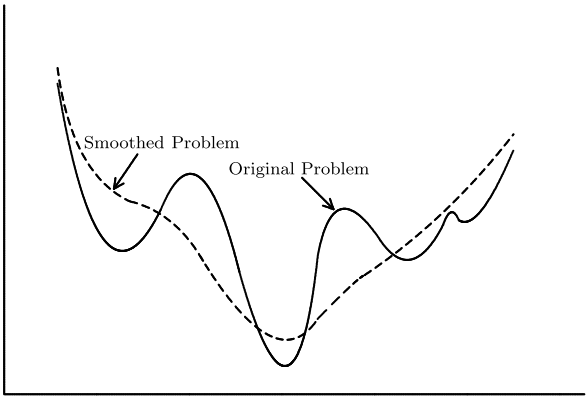

Fig. 1 Effect of smoothing the original optimization problem

where Φ is a strictly convex function. If Φ is chosen to have a Hessian that is *sufficiently positive definite* for all x, i.e., the eigenvalues of this Hessian are uniformly bounded away from zero, it implies that for μ large enough, $\mathcal{F}(x, \mu)$ is strictly convex. Similar results or proofs of such an assertion can be found, for example, in [1, Lemma 3.2.1].

Theorem 3.1 *Suppose* $f : [0, 1]^n \to \mathbb{R}$ *is a* C^2 *function and* $\Phi : X \to \mathbb{R}$ *is a* C^2 *function such that the minimum eigenvalue of* $\nabla^2\Phi(x)$ *is greater than or equal to* ϵ *for all* $x \in X$, *where* $X \subset [0, 1]^n$ *and* $\epsilon > 0$. *Then there exists* $M > 0$ *such that if* $\mu > M$, *then* $f + \mu\Phi$ *is a strictly convex function on* X.

Consequently, for μ sufficiently large, any local minimizer of $\mathcal{F}(x, \mu)$ is also the unique global minimizer. Typically the minimizer of $\Phi(x)$ is known or is easy to find and hence minimizing $\mathcal{F}(x, \mu)$ for large μ is also easy. This is important in smoothing methods because the basic idea is to solve the problem for a decreasing sequence of μ starting with a large value and ending with one that may be close to zero. The solution $x(\mu_k)$ of $\min_x \mathcal{F}(x, \mu_k)$ is used as the starting point of $\min_x \mathcal{F}(x, \mu_{k+1})$.

The idea behind global smoothing is similar to that of local smoothing, namely, the hope is that by adding $\mu\Phi(x)$ to $f(x)$, poor local minimizers will be eliminated. There are, however, important differences between global and local smoothing. A key one is that local smoothing does not guarantee that the function is unimodal for sufficiently large values of the smoothing parameter. For example, if the algorithm in [24] is applied to the function $\cos(x)$, a multiple of $\cos(x)$ is obtained. Thus, the number of minimizers of the smoothed function has not been reduced. It is easy to appreciate that the global smoothing approach is largely independent of the initial estimate of a solution, since if the initial function is unimodal, the choice of initial point is irrelevant to the minimizer found. When μ is decreased and the subsequent functions have several minimizers, the old solution is used as the initial point. Consequently, which minimizer is found is predetermined. Independence of the choice of initial point may be viewed as both a strength and a weakness. What is happening is that any initial point is being replaced by a point close to the minimizer of $\Phi(x)$. An obvious concern is that convergence will then be to the minimizer closest to the minimizer of $\Phi(x)$. The key to the success of this approach is to choose $\Phi(x)$ to have a minimizer

that is not close to any of the minimizers of $f(x)$. This may not seem easy, but it is possible for constrained problems. If it is known that the minimizers are on the edge of the feasible region (e.g., with concave objective functions), then the "center of the feasible region" may be viewed as being removed from all of them. An example is the analytic center and there are certainly many other choices of generating an initial feasible solution since the feasible region is linearly constrained.

4 Logarithmic smoothing

Consider the following relaxation of (1.1):

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & Ax = b, \\ & 0 \le x \le e. \end{array} \tag{4.1}$$

A suitable smoothing function that we have used is given by the *logarithmic smoothing* function:

$$\Phi(x) \equiv -\sum_{j=1}^{n} \ln x_j - \sum_{j=1}^{n} \ln(1 - x_j). \tag{4.2}$$

This function is clearly well-defined when $0 < x < e$. If any value of x_j is 0 or 1, we have $\Phi(x) = \infty$, which implies we can dispense with the bounds on x to get the following transformed problem:

$$\begin{array}{ll} \text{Minimize} & f(x) - \mu \sum_{j=1}^{n} [\ln x_j + \ln(1 - x_j)] \\ \text{subject to} & Ax = b, \end{array} \tag{4.3}$$

where $\mu > 0$ is the smoothing parameter. When a linesearch algorithm starts with an initial point $0 < x_0 < e$, then all iterates generated by the linesearch also satisfy this property, provided care is taken in the linesearch to ensure that the maximum step taken is within the bounds $0 < x < e$.

4.1 Properties of logarithmic smoothing function

The function $\Phi(x)$ is a *logarithmic barrier* function and is used with barrier methods (see [11]) to eliminate inequality constraints from a problem. In fact, (4.3) is sometimes known as the barrier subproblem for (4.1). Our use of this barrier function is not to eliminate the constraints but because a barrier function appears to be an ideal smoothing function. Elimination of the inequality constraints is a useful bonus. It also enables us to draw upon the extensive theoretical and practical results concerning barrier methods.

A key property of the barrier term is that $\Phi(x)$ is strictly convex. If μ is large enough, the function $f + \mu\Phi$ will also be strictly convex, as follows from Theorem 3.1. By Theorem 8 of [11], under the assumptions already imposed on (4.1),

if $x^*(\mu)$ is a solution to (4.3), then there exists a solution x^* to (4.1) such that $\lim_{\mu \searrow 0} x^*(\mu) = x^*$. The following theorem is then a consequence of the results of [11]:

Theorem 4.1 *Let $x(\mu, \gamma)$ be any local minimizer of*

$$\begin{array}{ll} \text{Minimize} & f(x) - \mu \sum_{j=1}^{n} [\ln x_j + \ln(1 - x_j)] + \gamma \sum_{j \in J} x_j(1 - x_j) \\ \text{subject to} & Ax = b, \\ & 0 \le x \le e. \end{array} \tag{4.4}$$

Then $\lim_{\gamma \to \infty} \lim_{\mu \searrow 0} x_j(\mu, \gamma) = 0$ or 1 for $j \in J$.

The general procedure of the barrier function method is to solve problem (4.3) approximately for a sequence of decreasing values of μ since [11] has shown that the solution to (4.3), $x^*(\mu)$, is a continuously differentiable curve. Note that if $\lim_{\mu \searrow 0} x^*(\mu) = x^* \in \{0, 1\}^n$, then we need not solve (4.3) for μ very small because the rounded solution for a modestly small value of μ should be adequate.

Consider now the example given in Sect. 2 to illustrate the possible failure of the use of a penalty function. The problem is now transformed to

$$\begin{array}{ll} \text{Minimize} & x^2 - \mu \log x - \mu \log(1 - x) + \gamma x(1 - x) \\ \text{subject to} & 0 \le x \le 1, \end{array} \tag{4.5}$$

where $\mu > 0$ is the barrier parameter. If $x^*(\mu, \gamma)$ denotes a stationary point then $x^*(10, 10) = 0.3588$ and $x^*(1, 10) = 0.1072$. Thus, rounding of $x^*(\mu, \gamma)$ in these cases will give the global minimizer. In fact, a trajectory of $x^*(\mu, 10)$ can be obtained such that $x^*(\mu, 10) \to 0$ as $\mu \searrow 0$.

We have in effect replaced the hard problem of a nonlinear integer optimization problem by what at first appearance is an equally hard problem of finding a global minimizer for a problem with continuous variables and a large number of local minima. The basis for our optimism that this is not the case lies in how we can utilize the parameters μ and γ and try to obtain a global minimizer, or at least a good local minimizer of the composite objective function. Note that the term $x_j(1 - x_j)$ attains its maximum at $x_j = \frac{1}{2}$ and that the logarithmic barrier term attains its minimum at the same point. Consequently, at this point, the gradient is given solely by $f(x)$. In other words, which vertex looks most attractive from the perspective of the objective is the direction we tilt regardless of the value of μ or γ. Starting at a neutral point and slowly imposing integrality is a key idea in the approach we advocate.

Also, note that provided μ is sufficiently large compared to γ, the problem will have a unique and hence global solution $x^*(\mu, \gamma)$, which is a continuous function of μ and γ. The hope is that the global or at least a good minimizer of (1.1) is the one connected by a continuous trajectory to $x^*(\mu, \gamma)$ for μ large and γ small.

Even if a global minimizer is not identified, we hope to obtain a good local minimizer and perhaps combine this approach with traditional methods.

4.2 The parameters μ and γ

The parameter μ is the standard barrier parameter. However, its initial value and how it is adjusted is not how this is typically done when utilizing the normal barrier function approach. For example, when a good estimate of a minimizer is known then in the normal algorithm one tries to choose a barrier parameter that is compatible with this point. By that we mean one that is suitably small. This is not required in the approach advocated here. Indeed it would be a poor policy. Since we are trying to find the global minimizer we do not want the iterates to converge to a local minimizer and it is vital that the iterates move away should the initial estimate be close to a local minimizer. The initial choice of μ is made to ensure it is suitably large. By that we mean it dominates the other terms in the objective. It is not difficult to do that since this does not incur a significant cost if it is larger than is necessary. The overall objective is trivial to optimize when μ is large. Moreover, subsequent objectives are easy to optimize when μ is reduced at a modest rate. Consequently, the additional effort of overestimating μ is small. Underestimating μ increases the danger of converging to a local minimizer. Unlike the regular barrier approach the parameter is reduced at a modest rate and a reasonable estimate is obtained to the solution of the current subproblem before the parameter is reduced. The basic idea is to stay close to the "central trajectory".

Even though there may not be a high cost to overestimating the initial value of μ it is sensible to make an attempt to estimate a reasonable value especially since this impacts the choice of γ. Generally, the initial value of μ can be set as the maximum eigenvalue of $\nabla^2 f(x)$ and it is easy to show that such a choice of μ would be sufficient to make the function $f + \mu\Phi$ strictly convex. If a poor initial value is chosen this can be deduced from the difference between the minimizer obtained and that of the minimizer of just the barrier function. This latter minimizer is easy to determine and may be used as the initial point for the first minimization. Another issue that differs from the regular barrier function method is that there is no need to drive μ to near zero before terminating. Consequently, the ill-conditioning that is typical when using barrier functions is avoided.

Estimating a suitable initial value for γ is more challenging. What is important is not just its value but its value relative to μ and $\|\nabla^2 F(x)\|$. The basic idea is that we start with a strongly convex function and as μ is decreased and γ is increased the function becomes nonconvex. However, it is important that the nonconvexity of the penalty term does not overwhelm the nonconvexity of $F(x)$ in the early stages, and so the initial value of γ should be much smaller than the absolute value of the minimum eigenvalue of $\nabla^2 F(x)$. Typically, an initial value of γ could be as small as 1% of the absolute value of the minimum eigenvalue of $\nabla^2 F(x)$. It helps that it is not necessary for $\gamma \to \infty$. Provided a good initial value is chosen the issue of adjusting it is not hard since it may be increased at a similar rate to that at which μ is decreased. Usually, θ_μ, the rate of decrease of μ, is set to be a value lying in the range of $[0.1, 0.9]$, while θ_γ, the rate of increase of γ, is set to be a value lying in the range of $[1.1, 10]$ with $\theta_\mu \theta_\gamma \approx 1$.

It is of interest to note what is the consequence of extreme strategies in the unconstrained case. If the initial values of μ and γ are small we will get a rounded solution

of the local minimizer that a typical active-set algorithm would converge from the initial point chosen. If the initial value of μ is large but that of γ is kept small except in the later stages, we will obtain the rounded solution of an attempt to find the global minimizer of $F(x)$. While this is better than finding the rounded solution of an arbitrary local minimizer it is not the best we can do. However, it is clear that of these two extremes the former is the worse. What is not known is how small the initial value of γ needs to be compared to the initial value of μ.

4.3 A simple example

To illustrate the approach, we show how it would work on the following two-variable problem. Let

$$f(x_1, x_2) = -(x_1 - 1)^2 - (x_2 - 1)^2 - 0.1(x_1 + x_2 - 2).$$

Consider the following problem

$$\begin{array}{ll} \text{Minimize} & f(x_1, x_2) \\ \text{subject to} & x_i \in \{0, 2\} \quad \text{for } i = 1, 2. \end{array} \tag{4.6}$$

Since f is separable, solving problem (4.6) is equivalent to solving two identical problems with one variable and a global optimal solution of $x_1^* = x_2^* = 2$ can easily be deduced.

In our approach, we first relax problem (4.6) to

$$\begin{array}{ll} \text{Minimize} & f(x_1, x_2) \\ \text{subject to} & 0 \le x_i \le 2 \quad \text{for } i = 1, 2. \end{array} \tag{4.7}$$

We can then transform (4.7) to the following unconstrained optimization problem by introducing logarithmic barrier terms:

$$\begin{aligned} \text{Minimize} \quad F(x_1, x_2) \equiv f(x_1, x_2) &- \mu[\log(x_1) + \log(2 - x_1) \\ &+ \log(x_2) + \log(2 - x_2)], \end{aligned} \tag{4.8}$$

where $\mu > 0$ and we have also omitted the implicit bound constraints on x_1 and x_2. The contours of F for 4 different values of μ are illustrated in Fig. 2.

Problem (4.8) can be solved for each μ by applying Newton's method to obtain a solution to the first-order necessary conditions of optimality for (4.8), i.e.,

$$\nabla F(x_1, x_2) = \begin{bmatrix} -2x_1 + 1.9 - \frac{\mu}{x_1} + \frac{\mu}{2 - x_1} \\ -2x_2 + 1.9 - \frac{\mu}{x_2} + \frac{\mu}{2 - x_2} \end{bmatrix} = 0.$$

Our approach involves solving (4.8) for a fixed $\mu = \mu_l > 0$ to obtain an iterate $x^{(l)} = (x_1^{(l)}, x_2^{(l)})$, and then using this iterate as the initial iterate to solve (4.8) for $\mu = \mu_{l+1}$, where $\mu_l > \mu_{l+1} > 0$. The initial iterate for the first μ parameter, μ_0, is set as $x^{(0)} = (1, 1)$, which is a point with equal Euclidean distances from the extreme points of the region $X = [0, 2] \times [0, 2]$. The resulting sequence of iterates forms a trajectory and

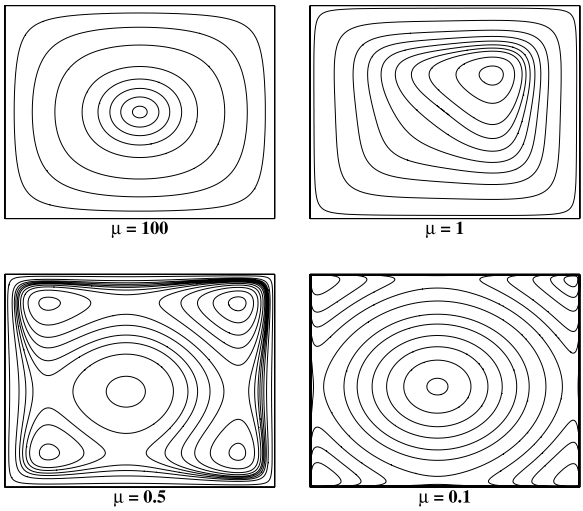

Fig. 2 The contours for the objective function of (4.8) as the parameter μ varies

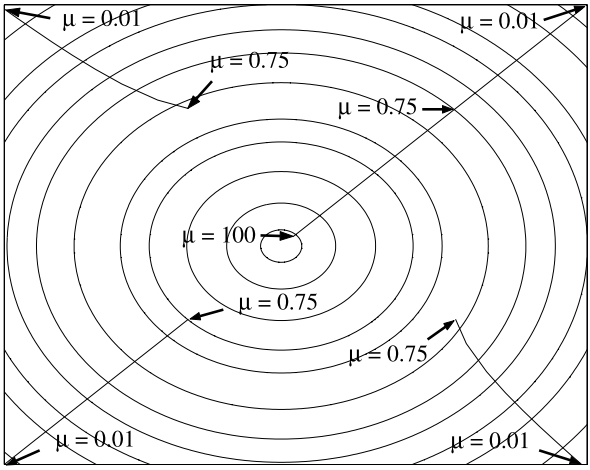

Fig. 3 Trajectories taken by iterates for problem (4.8)

converges to one of the extreme points of X. Fig. 3 illustrates the iterates as μ varies, as well as the possible trajectories. With the choice of a large μ_0 value of 100, we are able to obtain a trajectory that converges to the optimal solution of (4.6).

Though the above example is only a two-variable problem, our approach can easily be extended to problems with more variables. A point to note is that despite the local minimizers being quite similar both in terms of their objective function and the size of the corresponding sets S_v the transformed function is unimodal until μ is quite small and rounding at the solution just prior to the function becoming non-unimodal would give the global minimizer. Of course this is a simple example and it remains to be seen whether this feature is present for more complex problems.

5 Description of the algorithm

Since the continuous problems of interest may have many local minimizers and saddle points, first-order methods are inadequate as they are only assured of converging to points satisfying first-order optimality conditions. It is therefore imperative that second-order methods be used in the algorithm. Any second-order method that is assured of converging to a solution of the second-order optimality conditions must explicitly or implicitly compute a direction of negative curvature for the reduced Hessian matrix. A key feature of our approach is a very efficient second-order method for solving the continuous problem.

We may solve (4.4) for a specific choice of μ and γ by starting at a feasible point and generating a descent direction, if one exists, in the null space of A. Let Z be a matrix with columns that form a basis for the null space of A. Then $AZ = 0$ and the rank of Z is $n - m$. If x_0 is any feasible point so that we have $Ax_0 = b$, the feasible region can be described as $\{x : x = x_0 + Zy,\ \ y \in \mathbb{R}^{n-m}\}$. Also, if we let ϕ be the restriction of F to the feasible region, the problem becomes the following unconstrained problem:

$$\underset{y\in\mathbb{R}^{n-m}}{\text{Minimize}} \quad \phi(y). \tag{5.1}$$

Since the gradient of ϕ is $Z^{\mathrm{T}}\nabla F(x)$, it is straightforward to obtain a stationary point by solving the equation $Z^{\mathrm{T}}\nabla F(x) = 0$. This gradient is referred to as the *reduced gradient*. Likewise, the reduced Hessian, i.e., the Hessian of ϕ, is $Z^{\mathrm{T}}\nabla^2 F(x)Z$.

For small or moderately-sized problems, a variety of methods may be used (see e.g., [12, 15]). Here, we investigate the case where the number of variables is large. One approach to solving the problem is to use a linesearch method, such as the truncated-Newton method (see [8]) we are adopting, in which the descent direction and a direction of negative curvature are computed. Instead of using the index set J for the definition of F in our discussion, we let γ be a vector of penalty parameters with zero values for those x_i such that $i \notin J$, and $\Gamma = \mathrm{Diag}(\gamma)$.

The first-order optimality conditions for (4.4) (ignoring the implicit bounds on x) may be written as

$$\begin{aligned} &\nabla f - \mu X_g e + \Gamma(e - 2x) + A^{\mathrm{T}}\lambda = 0, \\ &Ax = b, \end{aligned} \tag{5.2}$$

where $X_g = \mathrm{Diag}(x_g)$, $(x_g)_i = \frac{1}{x_i} - \frac{1}{1-x_i}$, $i = 1, \ldots, n$, and λ corresponds to the Lagrange multiplier vector of the constraint $Ax = b$. Applying Newton's method directly, we obtain the system

$$\begin{bmatrix} H & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f + \mu X_g e - \Gamma\,(e - 2x) - A^{\mathrm{T}}\lambda \\ b - Ax \end{bmatrix}, \tag{5.3}$$

where $H = \nabla^2 f + \mu\ \mathrm{Diag}(x_H) - 2\Gamma$ and $(x_H)_i = \frac{1}{x_i^2} + \frac{1}{(1-x_i)^2}$, $i = 1, 2, \ldots, n$.

Assuming that x_0 satisfies $Ax_0 = b$, the second equation $A\triangle x = 0$ implies that $\triangle x = Zy$ for some y. Substituting this into the first equation and premultiplying both

sides by Z^{T}, we obtain

$$Z^{\mathrm{T}} H Z y = Z^{\mathrm{T}}(-\nabla f + \mu X_g e - \Gamma\,(e - 2x)). \tag{5.4}$$

To obtain a descent direction in this method, we first attempt to solve (5.4), or from the definition of $F(x)$, the equivalent reduced Hessian system

$$Z^{\mathrm{T}} \nabla^2 F(x_l) Z y = -Z^{\mathrm{T}} \nabla F(x_l), \tag{5.5}$$

by the conjugate gradient method, where x_l is the lth iterate. Generally, Z may be a large matrix, especially if the number of linear constraints is small. Thus, even though $\nabla^2 F(x_l)$ is likely to be a sparse matrix, $Z^{\mathrm{T}} \nabla^2 F(x_l) Z$ may be a large dense matrix. The virtue of the conjugate gradient method is that the explicit reduced Hessian need not be formed. There may be specific problems where the structure of $\nabla^2 F$ and Z does allow the matrix to be formed. Under such circumstances alternative methods such as those based on Cholesky factorization may also be applicable. Since we are interested in developing a method for general application we have pursued the conjugate gradient approach.

In the process of solving (5.5) with the conjugate gradient algorithm (see, [17]), we may determine that $Z^{\mathrm{T}} \nabla^2 F(x_l) Z$ is indefinite for some l. In such a case, we obtain a negative curvature direction q such that

$$q^{\mathrm{T}} Z^{\mathrm{T}} \nabla^2 F(x_l) Z q < 0.$$

This negative curvature direction is required to ensure that the iterates do not converge to a saddle point. Also, the objective is decreased along this direction. In practice, the best choice for q is an eigenvector corresponding to the smallest eigenvalue of $Z^{\mathrm{T}} \nabla^2 F(x_l) Z$. Computing q is usually expensive but fortunately unnecessary. A good direction of negative curvature will suffice and efficient ways of computing directions within a modified-Newton algorithm are described in [2]. The descent direction in such modified-Newton algorithms may also be obtained using factorization methods (see, e.g., [13]). In any case, it is essential to compute both a descent direction and a direction of negative curvature (when one exists). One possibility that may arise is that the conjugate gradient algorithm terminates with a direction q such that $q^{\mathrm{T}} Z^{\mathrm{T}} \nabla^2 F(x_l) Z q = 0$. In that case, we may have to use other iterative methods such as the Lanczos method to obtain a direction of negative curvature as described in [2]. The vector q is a good initial estimate to use in such methods.

If we let p be a suitable linear combination of the negative curvature direction q with a descent direction, the convergence of the iterates is still ensured with the search direction Zp. The next iterate x_{l+1} is thus defined by $x_l + \alpha_l Z p$, where α_l is being determined using a linesearch. The iterations will be performed until $Z^{\mathrm{T}} \nabla F(x_l)$ is sufficiently close to 0 and $Z^{\mathrm{T}} \nabla^2 F(x_l) Z$ is positive semidefinite. Also, as the current iterate x_l may still be some distance away from the actual optimal solution we are seeking, and since we do not necessarily use an exact solution of (5.5) to get the search direction, we only need to solve (5.5) approximately. An alternative is to use the two directions to form a differential equation (see [7]), which leads to a search along an arc.

A summary of the new algorithm is shown below:

Global Smoothing Algorithm: GSA

Set ϵ_F = tolerance for function evaluation,
ϵ_μ = tolerance for barrier value,
M = maximum penalty value,
N = iteration limit for applying Newton's method,
θ_μ = ratio for barrier parameter decrement,
θ_γ = ratio for penalty parameter increment,
μ_0 = initial barrier parameter,
γ_0 = initial penalty parameter,
r = any feasible starting point.
Set $\gamma = \gamma_0$, $\mu = \mu_0$,
while $\gamma < M$ or $\mu > \epsilon_\mu$.
Set $x_0 = r$
for $l = 0, 1, \ldots, N$,
if $\|Z^{\mathrm{T}} \nabla F(x_l)\| < \epsilon_F \mu$,
Set $x_N = x_l$, $l = N$.
Check if x_N is a direction of negative curvature,
else
Apply conjugate gradient algorithm to
$[Z^{\mathrm{T}} \nabla^2 F(x_l) Z] y = -Z^{\mathrm{T}} \nabla F(x_l)$.
Obtain p_l as a combination of directions of descent and negative curvature.
Perform a linesearch to determine α_l and set
$x_{l+1} = x_l + \alpha_l Z p_l$,
end if
end for
Set $r = x_N$, $\mu = \theta_\mu \mu$, $\gamma = \theta_\gamma \gamma$.
end while

6 Computational results

To show the behavior of GSA we give some results on problems in which problem size can be varied. The purpose of these results is more for illustration than as conclusive evidence of the efficacy of the new algorithm. Still the success achieved suggests that the algorithm has merit. More results are given in [26] and what these results here show is that there are problems for which GSA performs well. Moreover, the rate of increase in computational work as problem size increases is modest. All results for GSA were obtained from a MATLAB 6.5 implementation on an Intel Pentium IV 2.4 GHz PC with 512 MB RAM running on Windows XP Operating System.

As a preliminary illustration of the performance of GSA, we consider the following set of test problems from [28]:

$$\begin{aligned} &\text{Minimize} \quad -(n-1)\sum_{i=1}^{n} x_i - \frac{1}{n}\sum_{i=1}^{n/2} x_i + 2 \sum_{1 \le i < j \le n} x_i x_j \\ &\text{subject to} \quad x_i \in \{0, 1\} \quad \text{for } i = 1, 2, \ldots, n. \end{aligned} \tag{6.1}$$

In the case when n is even, Pardalos [28] shows that the unique global minimum of (6.1) is given by the feasible point

$$x^* = (\underbrace{1, \ldots, 1}_{n/2}, \underbrace{0, \ldots, 0}_{n/2}),$$

with objective value $-\frac{n^2+2}{4}$, and there is an exponential number of discrete local minima for (6.1). Here, a point $x \in \{0, 1\}^n$ is a discrete local minimum if and only if $f(x) \leq f(y)$ for all points $y \in \{0, 1\}^n$ adjacent to x, when all points are considered to be vertices of the unit hypercube $\{0, 1\}^n$.

As $\nabla^2 f(x) = 2(ee^{\mathrm{T}} - I)$ for this set of test problems, it is possible to simplify all the matrix-vector products involving $\nabla^2 f(x)$ in GSA by taking note that $[\nabla^2 f(x)]v = 2[(e^{\mathrm{T}}v)e - v]$ for any $v \in \mathbb{R}^n$, leading to more efficient computations. The initial μ and γ parameters are set as 100 and 0.1 respectively, with a μ-reduction ratio of 0.1 and γ-increment ratio of 10. The analytic center of $[0, 1]^n$, i.e., $\frac{1}{2}e$, is used as the initial iterate and the tolerance is set as $\epsilon_\mu = 10^{-3}$ for the termination criteria. In all the instances tested with the number of variables ranging from 1000 to 10000, GSA was able to obtain convergence to the global optimal solution. In particular, it is possible to show that using the initial iterate of $\frac{1}{2}e$, the next iterate obtained by GSA would be of the form $(\underbrace{u_1, \ldots, u_1}_{n/2}, \underbrace{u_2, \ldots, u_2}_{n/2})$ where $u_1 > \frac{1}{2}$ and $u_2 < \frac{1}{2}$, i.e., an immediate rounding of this iterate would have given the global optimal solution.

A plot of the computation time required versus the number of variables is shown in Fig. 4. These results indicate that the amount of computational work in GSA does not increase drastically with an increase in problem size for this set of test problems. Moreover, it illustrates the ability of GSA to find the optimal solution to large problems with as much as 10000 variables within a reasonable amount of computation time, as well as the potential to exploit possible problem structure to improve computational efficiency in solving problems.

For purposes of evaluating the performance of GSA especially when we do not know the exact optimal solutions to the test problems, we have used some of the nonlinear mixed-integer programming solvers of GAMS,[1] namely DICOPT, SBB and BARON as described earlier. MINOS was used as the common underlying nonlinear programming solver required by all the three solvers, while CPLEX was used as the underlying linear programming solver required by BARON. While DICOPT and SBB solvers are able to handle nonlinear discrete optimization problems, these solvers do not necessarily attain the global optimal solution for nonconvex problems. As BARON is a global optimization solver that could potentially return the exact optimal solution or relevant bounds on the optimal objective value, we have made extensive comparisons between GSA and BARON in certain computational instances. We have used GAMS version 22.2 that includes BARON version 7.5.

When using the GAMS solvers, other than the maximum computation time, the maximum number of iterations and the maximum amount of workspace memory allocated to the solvers, the other options were set to default values. We generally

[1] http://www.gams.com.

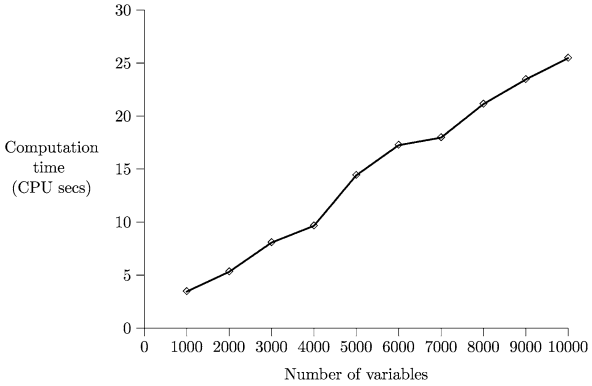

Fig. 4 Graph of computation time required (CPU seconds) for solving instances of problem (6.1)

set a computation time limit of 10000 CPU seconds and the maximum number of iterations to 500 000 for the runs. To ensure that the solvers have a reasonable amount of memory for setting up problem instances to solve, we have configured a maximum workspace memory of 1 GB. It turns out that less than 100 MB were actually needed by all the GAMS solvers to set up each of the problem instances.

All the runs involving the GAMS solvers were mostly performed using the same PC used for GSA, except where noted. It should be pointed out that GSA requires very little additional memory other than that needed to define the problem, which is a tiny fraction of that used by the other methods considered here.

6.1 Unconstrained binary quadratic problems

One of the simpler classes of problems in the form of (1.1) with a nonlinear objective function is that of the unconstrained binary quadratic problem, i.e., the objective function is quadratic in the binary variables x, and both A and b are the zero matrix and the zero vector respectively. There are many relaxation and approximation approaches available for this class of problems, such as [4] and [21]. Despite its simplicity such problems are still hard to solve as the numerical testing reveals. What these problems give is another opportunity to examine performance as size increases even though such a numerical comparison may not be sufficient to measure the efficacy of GSA for other types of nonlinear optimization problems with binary variables. Moreover, there exists publicly available test problems in this class of optimization problems.

The unconstrained binary quadratic problem (BQP) may be stated as follows:

$$\begin{array}{ll} \text{Minimize} & x^{\mathrm{T}} P x + c^{\mathrm{T}} x \\ \text{subject to} & x \in \{0,1\}^n, \end{array} \tag{6.2}$$

where $P \in \mathbb{R}^{n\times n}$ and $c \in \mathbb{R}^n$. By noting that $x^{\mathrm{T}} M x = x^{\mathrm{T}} M^{\mathrm{T}} x$ and $c^{\mathrm{T}} x = x^{\mathrm{T}} C x$ for any feasible x and any $M \in \mathbb{R}^{n\times n}$, where $C = \mathrm{Diag}(c)$, it suffices to consider the following "simpler" problem

$$\begin{array}{ll} \text{Minimize} & x^{\mathrm{T}} Q x \\ \text{subject to} & x \in \{0,1\}^n, \end{array} \tag{6.3}$$

where Q is a symmetric matrix.

Since any unconstrained quadratic programming problem with purely integer variables from a bounded set can be transformed into (6.2), many classes of problems would then fall under this category of BQP problems, such as the least-squares problem with bounded integer variables:

$$\underset{x \in D}{\text{Minimize}} \quad \|s - Ax\|,$$

where $s \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, and D is a bounded subset of $\mathbb{Z}^n$.

A test set for BQP problems in the form of (6.3) is given in the OR-Library, which is maintained by J.E. Beasley.[2] The entries of matrix Q are integers uniformly drawn from $[-100, 100]$, with density 10%. As the test problems were formulated as maximization problems, for purposes of comparison of objective values, we now maximize the objective function.

The GSA was run on all 60 problems of Beasley's test set ranging from 50 to 2500 variables. The initial μ and γ parameters are set as 100 and 1 respectively, with a μ-reduction ratio of 0.5 and γ-increment ratio of 2. The analytic center of $[0, 1]^n$, i.e., $\frac{1}{2}e$, is used as the initial iterate and the tolerance is set as $\epsilon_\mu = 0.1$ for the termination criteria. For the GAMS solvers, we initially notice that DICOPT and SBB always return the zero vector as the solution to the test problems, which could be due to the solvers accepting a local solution in the underlying nonlinear programming subproblems being solved. To allow these two solvers to return non-zero solutions for comparison, we have added the cut $e^{\mathrm{T}}x \geq 1$ or $e^{\mathrm{T}}x \geq 2$ when applying DICOPT and SBB on the test problems. The objective values obtained by the GSA and the respective GAMS solvers, as well as the respective computation time and number of iterations required are given in Tables 1–3. There are preprocessing procedures in BARON that attempt to obtain a good solution before letting the solver find the optimal or a better solution and the results obtained by these preprocessing procedures are shown in the tables. Finally, the objective values of the best known solutions based on RUTCOR's website[3] are given in the last column of the tables.

BARON found the optimal solution for all 20 problems in the test sets involving 50 to 100 binary variables, with the objective values listed in Tables 1(a), (b). Using these optimal objective values as a basis of comparison, we conclude that GSA successfully found 11 of these optima, with another 6 solutions obtained having less than 0.5% deviation from the optimal objective value. The percentage deviation in the remaining 3 solutions obtained by the GSA were also close, with the worst one having 1.66% deviation from the optimal objective value. On the other hand, DICOPT obtained results ranging from 2.25% to 26.26% deviation from the optimal objective value for the 50-variable test problems and from 0.56% to 13.03% for the 100-variable test problems, while SBB obtained results ranging from 1.9% to 26.26% deviation from the optimal objective value for the 50-variable test problems and from 0.56% to 13.03% for the 100-variable test problems. For all these instances, GSA was able to obtain better objective values than DICOPT or SBB.

[2] http://mscmga.ms.ic.ac.uk/info.html.

[3] http://rutcor.rutgers.edu/~pbo/families/beasley.htm.

Table 1 Numerical comparison of algorithms applied to Beasley's BQP test problems (maximization objective) with (a) 50 binary variables; (b) 100 binary variables

Problem instance	DICOPT			SBB			BARON						GSA				Best known obj. value
							Preprocessing		Postprocessing								
	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Obj. value	Time (sec)	Obj. value	Upper bound	Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count	
(a)																	
50_1	1547	0.03	18	1547	0.38	18	2030	0.08	2098	2098	0.09	1	2098	0.28	1.00	119	2098
50_2	3360	0.98	27	3375	0.17	27	3678	0.08	3702	3702	0.12	1	3702	0.22	0.86	105	3702
50_3	4463	0.05	35	4463	0.05	35	4626	0.08	4626	4626	0.09	1	4626	0.61	1.61	114	4626
50_4	3412	0.36	62	3415	0.28	30	3544	0.11	3544	3544	0.16	1	3544	0.34	1.06	111	3544
50_5	3796	0.03	31	3796	0.05	31	4012	0.08	4012	4012	0.11	1	4012	0.22	0.67	113	4012
50_6	3610	0.98	162	3610	0.42	46	3659	0.12	3693	3693	0.27	1	3693	0.25	0.61	109	3693
50_7	4379	0.03	36	4379	0.03	36	4520	0.08	4520	4520	0.09	1	4510	0.06	1.06	114	4520
50_8	4016	0.37	79	4136	0.20	41	4216	0.07	4216	4216	0.14	1	4216	0.27	0.72	113	4216
50_9	3370	0.05	35	3370	0.04	35	3780	0.11	3780	3780	0.25	1	3732	0.36	0.80	115	3780
50_10	3047	0.57	137	3047	0.21	31	3505	0.12	3507	3507	0.34	1	3505	0.58	0.91	114	3507
(b)																	
100_1	7218	0.04	63	7218	0.08	63	7792	0.12	7970	7970	949.86	1527	7838	0.84	1.33	128	7970
100_2	10922	2.01	262	10922	0.38	67	11026	0.13	11036	11036	14.95	35	10986	0.49	1.17	125	11036
100_3	12652	0.92	224	12652	0.43	74	12723	0.16	12723	12723	10.38	9	12723	0.34	1.13	127	12723
100_4	10136	0.08	65	10136	0.13	65	10368	0.10	10368	10368	46.70	85	10368	0.22	1.06	128	10368
100_5	8664	0.06	60	8664	0.06	60	8916	0.14	9083	9083	56.16	75	9040	0.58	1.58	155	9083
100_6	9579	0.86	189	9586	0.34	68	10126	0.14	10210	10210	805.09	1787	10101	0.30	1.30	144	10210
100_7	9676	0.06	63	9676	0.19	63	10035	0.14	10125	10125	60.50	135	10094	0.91	1.53	140	10125
100_8	10966	0.05	56	10966	0.10	56	11380	0.08	11435	11435	15.30	41	11435	0.52	1.00	124	11435
100_9	9962	0.06	63	9962	0.07	63	11455	0.10	11455	11455	7.94	11	11455	0.22	1.11	131	11455
100_10	12027	0.79	206	12027	0.25	73	12547	0.17	12565	12565	9.92	25	12547	0.41	1.25	123	12565

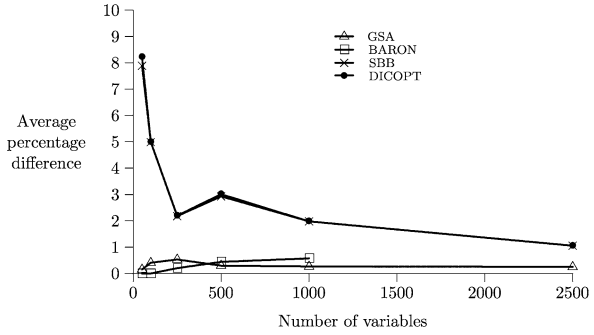

Fig. 5 Graph of average percentage difference in objective value of solution obtained by the various algorithms to best known solution when solving Beasley's BQP test problems

For the test sets with more than 100 variables, BARON was not able to complete the runs by the computation time limit of 10000 seconds and so the runs were truncated with BARON returning the best found solution. Thus, we do not have optimal objective values as a reference for these larger test problems, except for the upper bounds obtained by BARON or the objective values obtained from the RUTCOR website. We were unable to obtain any solution from BARON for the 2500-variable test problems within the computation time limit and this is also the case despite trying out on another platform, which is an Intel Pentium IV 2.4 GHz server with 2 GB RAM running on Linux Operating System.

From Tables 2 and 3, we can see that GSA again produced better solutions than DICOPT and SBB in all the instances. For all the test problems with 250 variables, GSA obtained objective values that have less than 2% deviation from the best-found objective value, with the worst one being 1.55% deviation. For the test problems with 500 variables, GSA obtained objective values that are better than BARON in 7 out of 10 instances, with all the instances having less than 0.64% deviation from the best-found objective value. For the test problems with 1000 variables, GSA obtained objective values that are better than BARON in 9 out of 10 instances, with the remaining instance having 0.62% deviation from the best-found objective value, which is also the worst deviation out of the 10 instances. For the test problems with 2500 variables, GSA obtained objective values ranging from 0.09% to 0.43% deviation from the best-found objective value. A graph showing the average percentage difference in the objective value of the solution obtained by the GSA and the GAMS solvers to that of the best known solution for Beasley's test problems is given in Fig. 5.

Figure 6 gives a plot of the relative average computation time required to solve the Beasley's test problems by DICOPT, SBB, BARON and GSA, with respect to problem size. Since we were unable to determine the computation time needed by BARON to complete its run for test problems of certain sizes, there were only limited data points for BARON in the figure. It can be seen that while there is a substantial increase in computation time required by BARON to solve larger test problems, the increase in computation time for the other solvers and GSA has been less significant. However, GSA has been able to maintain the quality of solutions obtained when compared to all the three solvers for problems with increasing size as illustrated in Fig. 5, indicating its efficiency and potential in obtaining good solutions to large problems without incurring unreasonable computation times.

Table 2 Numerical comparison of algorithms applied to Beasley's BQP test problems (maximization objective) with (a) 250 binary variables; (b) 500 binary variables

Problem instance	DICOPT			SBB			BARON						GSA				Best known obj. value
							Preprocessing		Postprocessing								
	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Obj. value	Time (sec)	Obj. value	Upper bound	Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count	
(a)																	
250_1	45079	1.65	471	45079	0.50	164	45453	1.81	45579	74589.8	>1e4	169	45463	2.92	7.94	169	45607
250_2	43117	0.28	178	43117	0.30	178	44149	1.64	44438	74902.3	>1e4	193	44251	1.36	6.30	150	44810
250_3	48326	0.29	188	48326	0.27	188	48947	2.09	49037	77808.8	>1e4	175	48947	4.81	8.74	161	49037
250_4	41127	0.33	174	41127	0.28	174	40837	2.03	41219	71448.0	>1e4	180	41181	2.30	5.20	147	41274
250_5	47471	0.28	180	47471	0.28	180	47928	1.81	47955	75927.8	>1e4	190	47845	4.05	7.56	147	47961
250_6	39502	1.14	304	39502	0.47	159	40593	1.64	40777	74665.0	>1e4	198	40797	1.84	5.95	145	41014
250_7	46323	0.30	183	46323	0.28	183	46757	1.75	46757	76411.5	>1e4	199	46757	2.25	5.41	144	46757
250_8	34037	1.53	467	34037	0.67	137	34545	1.65	35650	68558.5	>1e4	225	35174	3.81	7.53	159	35726
250_9	48048	1.01	360	48048	0.80	181	48690	1.70	48813	77998.3	>1e4	208	48705	2.00	5.64	146	48916
250_10	39194	1.52	457	39264	0.48	158	40140	1.61	40442	72113.3	>1e4	221	40198	3.59	6.80	155	40442
(b)																	
500_1	114182	1.52	356	114182	1.50	356	115141	28.39	115141	308407.3	>1e4	7	115848	16.77	26.89	166	116586
500_2	125191	1.56	357	125191	1.42	357	128085	28.23	128085	307310.5	>1e4	14	128001	15.84	26.56	159	128339
500_3	128512	1.52	358	128512	1.48	358	130033	27.78	130547	314995.3	>1e4	10	130812	11.80	18.17	148	130812
500_4	125445	2.53	662	126536	1.91	378	128962	27.46	129298	313618.8	>1e4	13	129647	11.44	31.53	184	130097
500_5	123066	1.40	325	123066	1.33	325	124785	26.63	124785	309763.8	>1e4	14	125141	9.67	18.55	156	125487
500_6	120863	1.54	368	120863	1.52	368	120258	26.77	121074	308183.5	>1e4	13	121603	12.89	29.08	174	121772
500_7	117619	1.39	326	117619	1.41	326	121748	24.89	122164	309975.5	>1e4	10	121872	4.66	14.80	165	122201
500_8	118755	1.55	378	118755	1.59	378	122657	25.47	123282	311251.3	>1e4	13	123329	14.06	26.39	176	123559
500_9	115548	1.41	336	115548	1.36	336	119518	25.13	119921	309643.3	>1e4	14	120456	12.84	23.05	162	120798
500_10	123266	1.42	342	123266	1.50	342	129963	25.15	130547	314725.0	>1e4	14	129849	14.81	25.36	166	130619

Table 3 Numerical comparison of algorithms applied to Beasley's BQP test problems (maximization objective) with (a) 1000 binary variables; (b) 2500 binary variables

Problem instance	DICOPT			SBB			BARON						GSA				Best known obj. value
							Preprocessing		Postprocessing								
	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Obj. value	Time (sec)	Obj. value	Upper bound	Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count	
(a)																	
1000_1	361439	13.76	1831	361439	19.63	725	370209	399.52	370513	1257155.8	>1e4	1	370926	37.08	84.78	180	371438
1000_2	346137	16.56	1841	346153	12.20	706	350619	400.77	352816	1252151.0	>1e4	1	353621	41.72	84.44	167	354932
1000_3	364005	14.25	1776	364009	11.81	716	368936	374.87	369864	1264529.8	>1e4	1	369972	31.86	83.73	177	371236
1000_4	369083	13.40	751	369083	11.63	751	368356	360.27	368770	1270025.0	>1e4	1	369894	31.80	67.06	169	370675
1000_5	342971	12.52	705	342971	10.84	705	348893	369.69	349774	1261032.5	>1e4	1	352358	42.25	83.00	202	352760
1000_6	353908	11.50	680	353908	10.48	680	357013	369.39	357610	1257987.3	>1e4	1	358578	48.20	73.42	175	359629
1000_7	363538	11.36	686	363538	10.58	686	369342	365.88	369342	1259804.3	>1e4	1	370484	71.83	107.25	208	371193
1000_8	345011	11.56	716	345011	10.88	716	348733	365.66	348871	1253878.0	>1e4	1	350762	74.97	106.09	186	351994
1000_9	343910	11.53	753	343910	11.56	753	346534	364.28	346761	1255519.3	>1e4	1	349187	45.53	71.75	185	349337
1000_10	343337	12.34	1823	343337	11.72	730	347663	354.94	349717	1241158.5	>1e4	1	349222	35.34	90.88	176	351415
(b)																	
2500_1	1493833	226.8	1851	1493833	166.2	1851	–	>1e4	–	–	>1e4	–	1512053	576.8	788.98	325	1515944
2500_2	1454836	347.0	3100	1454870	170.5	1880	–	>1e4	–	–	>1e4	–	1467628	481.1	638.1	251	1471392
2500_3	1396881	185.9	5713	1396881	178.0	1940	–	>1e4	–	–	>1e4	–	1408480	786.1	926.6	243	1414192
2500_4	1490067	332.4	1909	1490067	171.2	1909	–	>1e4	–	–	>1e4	–	1503939	425.4	572.7	223	1507701
2500_5	1481727	178.0	1985	1481727	177.5	1985	–	>1e4	–	–	>1e4	–	1490179	357.6	669.2	274	1491816
2500_6	1460909	293.6	1981	1460909	176.7	1981	–	>1e4	–	–	>1e4	–	1466633	503.3	687.5	216	1469162
2500_7	1463076	322.0	3188	1463076	183.6	2020	–	>1e4	–	–	>1e4	–	1472716	400.2	634.8	266	1479040
2500_8	1470013	169.1	1893	1470013	170.0	1893	–	>1e4	–	–	>1e4	–	1482859	365.9	738.1	242	1484199
2500_9	1460611	185.5	1935	1460611	173.7	1935	–	>1e4	–	–	>1e4	–	1480494	516.7	714.2	241	1482413
2500_10	1470882	175.5	1948	1470882	174.7	1948	–	>1e4	–	–	>1e4	–	1477279	354.7	569.6	252	1483355

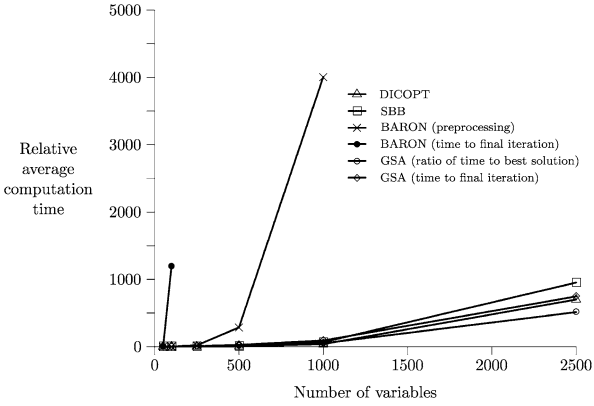

Fig. 6 Graph of relative average computation time required by the algorithms for solving Beasley's BQP test problems

As the number of local minima for the problems of Beasley's test set may be small, we also applied GSA to problems generated from the standardized test problem generator (Subroutine Q01MKD) written by P.M. Pardalos in [28], which may have a larger expected number of local minima. In generating the test matrices, we set the density of the matrices to be 0.1, with all entries of the matrices being integers bounded between -100 and 100. The seed to initialize the random number generator was set to 1. Note that unlike Beasley's test set, the test problems were formulated in [28] as minimization problems.

The GSA was run on these test problems ranging from 200 to 1000 binary variables. We used the same parameters as in Beasley's test problems, i.e., initial μ and γ being 100 and 1 respectively with a μ-reduction ratio of 0.5 and γ-increment ratio of 2. The analytic center of $[0, 1]^n$, i.e., $\frac{1}{2}e$, is also used as the initial iterate and a tolerance of $\epsilon_\mu = 0.1$ is used for the termination criteria. The objective values obtained by the GSA and the three GAMS solvers, as well as the respective computation time and number of iterations required are given in Table 4. The DICOPT and SBB results were obtained after adding the cut $e^{\mathrm{T}}x \geq 1$ as was done in the Beasley's test sets. However, we found that while DICOPT and SBB were able to return solutions to all the test problems, BARON was unable to return any solution for test problems with more than 700 variables. As such, we have performed all the BARON runs on the Linux platform mentioned earlier and BARON was then able to return the solutions reported in Table 4 for all the test problems.

From Table 4, we can see that GSA still produced better solutions than DICOPT and SBB in all the 17 instances. GSA also obtained objective values that are better than BARON in 13 instances. For the remaining 4 instances, other than one instance in which GSA obtained the same objective value as BARON, the percentage deviations from the best found solution with respect to objective values are all less than 0.25%. Thus the solutions obtained by GSA for the remaining instances are also comparable to that of BARON.

6.2 Other types of nonlinear optimization problems

The objective functions considered so far are relatively simple being purely quadratic functions and could be solved by specific methods for such problems, such as [27].

Table 4 Numerical comparison of algorithms for solving Pardalos's BQP test problems (minimization objective) with 200 to 1000 binary variables

Problem size	DICOPT			SBB			BARON						GSA			
	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Preprocessing		Postprocessing		Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count
							Obj. value	Time (sec)	Obj. value	Lower bound						
200	−34913	3.6	709	−34913	2.4	179	−35404	2.1	−35408	−4.8e4	>1e4	341	−35408	1.8	5.3	141
250	−44394	5.7	1518	−45289	2.5	235	−45170	4.8	−45475	−7.6e4	>1e4	118	−45516	8.0	13.6	189
300	−54526	1.5	431	−55448	3.5	241	−56292	10.7	−56936	−1.1e5	>1e4	51	−56966	5.0	11.9	154
350	−69099	5.1	1460	−69594	3.6	279	−69897	18.6	−70225	−1.5e5	>1e4	26	−70625	9.7	19.8	166
400	−80668	5.3	1584	−80772	2.6	283	−84026	32.1	−84786	−2.0e5	>1e4	15	−85448	11.4	23.7	186
450	−111534	3.2	605	−111534	3.4	320	−112087	51.0	−112340	−2.6e5	>1e4	7	−112790	25.9	42.3	190
500	−123982	5.1	1307	−124348	5.1	340	−129212	75.2	−129444	−3.2e5	>1e4	5	−129852	15.6	28.5	160
550	−141026	6.4	1105	−144857	7.4	478	−145008	107.6	−145405	−3.9e5	>1e4	5	−145565	24.6	42.8	171
600	−150479	7.4	1209	−151236	12.0	454	−154702	151.4	−155753	−4.6e5	>1e4	1	−155615	38.0	55.5	210
650	−188059	7.8	907	−190737	33.7	553	−192353	206.9	−192353	−5.4e5	>1e4	1	−192902	22.8	49.1	179
700	−214125	9.2	964	−214181	27.7	549	−216110	270.6	−217739	−6.2e5	>1e4	1	−217628	41.0	74.0	179
750	−221304	12.3	2099	−223495	30.5	596	−227174	355.1	−227913	−7.1e5	>1e4	1	−228209	34.4	71.7	171
800	−243821	14.8	2561	−245284	32.3	613	−245886	455.6	−247023	−8.1e5	>1e4	1	−248328	58.1	92.1	223
850	−266907	16.7	1143	−267325	26.3	683	−268361	577.2	−269295	−9.2e5	>1e4	1	−269945	51.3	99.8	194
900	−276661	21.0	1800	−276718	39.2	650	−283977	722.9	−287218	−1.0e6	>1e4	1	−288014	99.3	158.0	225
950	−318730	23.7	2527	−318751	41.3	658	−326427	891.4	−326427	−1.2e6	>1e4	1	−325640	72.0	148.9	197
1000	−331726	22.4	1292	−331940	35.4	695	−334556	1086.1	−337459	−1.3e6	>1e4	1	−338797	64.7	126.0	206

It may be seen that when BARON works well the preprocessor in the BARON algorithm often obtains good solutions. That is unlikely to be the case for real problems especially when continuous variables are present and appear nonlinearly. We now apply GSA to test problems with nonlinear nonquadratic objective functions. The first one considered is a quartic function generated from the product of two quadratic functions. For ease in deriving the optimal solution, we have used the following matrix Q mentioned in Example 4 of [28] for our construction of the test problems:

$$Q = \begin{bmatrix} 15 & -4 & 1 & 0 & 2 \\ -4 & -17 & 2 & 1 & 1 \\ 1 & 2 & -25 & -8 & 1 \\ 0 & 1 & -8 & 30 & -5 \\ 2 & 1 & 1 & -5 & -20 \end{bmatrix}.$$

In that example, we know that the solution to problem (6.3) with this Q matrix is given by $x^* = (0, 1, 1, 0, 1)^{\mathrm{T}}$ with objective value -54.

We define the following $(5n \times 5n)$-block diagonal matrix and $(5n \times 1)$-vector:

$$Q_n = \begin{bmatrix} Q & 0 & \dots & 0 \\ 0 & Q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Q \end{bmatrix},$$

$$x_n^* = \left((x^*)^{\mathrm{T}}(x^*)^{\mathrm{T}} \dots (x^*)^{\mathrm{T}}\right)^{\mathrm{T}}.$$

Then it is easy to see that an optimal solution to problem (6.3) with $Q = Q_n$ is given by $x = x_n^*$ with objective value $-54n$.

The test problems can now be defined in the general form as follows:

$$\begin{array}{ll} \text{Minimize} & (x^{\mathrm{T}} Q_n x + 54n + 1)(y^{\mathrm{T}} Q_n y + 54n + 1) \\ \text{subject to} & x, y \in \{0, 1\}^{5n}. \end{array} \tag{6.4}$$

The rationale for constructing these test problems in the above form, especially in the addition of a constant to each quadratic term in the product, is evident from the following observation:

Proposition 6.1 *An optimal solution to problem* (6.4) *is given by* $x = y = x_n^*$ *with objective value* 1.

Proof From above, for any $x \in \{0, 1\}^{5n}$, $x^{\mathrm{T}} Q_n x \geq -54n$ with equality achieved when $x = x_n^*$. Then we have $x^{\mathrm{T}} Q_n x + 54n + 1 \geq 1$ and $y^{\mathrm{T}} Q_n y + 54n + 1 \geq 1$ for any $x, y \in \{0, 1\}^{5n}$ with equality achieved when $x = x_n^*$ and $y = x_n^*$ respectively. Thus, $(x^{\mathrm{T}} Q_n x + 54n + 1)(y^{\mathrm{T}} Q_n y + 54n + 1) \geq 1$ for any $x, y \in \{0, 1\}^{5n}$ with equality achieved when $x = y = x_n^*$. □

Both GSA and the GAMS solvers were applied to this set of test problems with n ranging from 1 to 200, i.e., the number of binary variables ranges from 10 to 2000.

More experimentation was being carried out for test problems with smaller problem sizes as we detected a deterioration in the performance of the GAMS solvers for small test problems. The initial μ and γ parameters of the GSA are set as 100 and 1 respectively, with a μ-reduction ratio of 0.5 and γ-increment ratio of 2. The analytic center of $[0, 1]^n$, i.e., $\frac{1}{2}e$, is used as the initial iterate and the tolerance is set as $\epsilon_\mu = 0.1$ for the termination criteria. As we found that DICOPT and SBB are again returning the zero vector as the solution, we have also imposed the cut $e^{\mathrm{T}}x + e^{\mathrm{T}}y \geq 1$ for the DICOPT and SBB runs. The objective values obtained by the GSA and the three GAMS solvers, as well as the respective computation time and number of iterations required are given in Table 5.

GSA found the optimal solution for all the 14 instances and the computation time required to reach the optimal solution is less than 30 seconds for each instance, with almost all the instances requiring less than 10 seconds. BARON was able to find the optimal solution to the first 3 instances, with a significant increase in computation time even for a small instance of 30 variables. The objective values obtained by BARON for the other 11 instances with size more than 30 variables have not been satisfactory. The lower bounds found by BARON are also poor for these larger instances and are hardly helpful. DICOPT and SBB were unable to find any optimal solution to all the instances.

In the test problems we have considered so far, the objective function involves polynomial terms only. To show that GSA is able to handle other types of differentiable objective functions, we have constructed another set of test problems that includes a sum of exponential functions in the objective function. The objective function constructed here follows closely to that of problem (6.1). Also, the previous test problems do not show GSA's performance on test problems with constraints. Thus, we have added a linear constraint in this set of test problems, which also enables an easier argument for deriving the optimal solution.

The test problems can be defined as follows:

$$\begin{aligned} \text{Minimize} \quad & -(n-1)\sum_{i=1}^{n} x_i - \frac{1}{n}\sum_{i=1}^{n/2} x_i + 2\sum_{1\leq i<j\leq n} e^{x_i x_j} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i \leq \frac{n}{2}, \\ & x_i \in \{0, 1\} \quad \text{for } i = 1, 2, \ldots, n, \end{aligned} \tag{6.5}$$

where n is even. By extending the argument in [28], we have the following result for this set of test problems:

Proposition 6.2 *$x^* = (\underbrace{1, \ldots, 1}_{p^*}, \underbrace{0, \ldots, 0}_{n-p^*})$ is an optimal solution to problem (6.5) with objective value*

$$(e-1)(p^*)^2 + \left(2 - n - \frac{1}{n} - e\right)p^* + n^2 - n, \quad \textit{where } p^* = \left[\frac{n + \frac{1}{n} + e - 2}{2(e-1)}\right]$$

and $[x]$ refers to the integer obtained from rounding x.

Table 5 Numerical comparison of algorithms for solving test problems (6.4) (minimization objective) with 10 to 2000 variables

Problem size	DICOPT			SBB			BARON						GSA			
							Preprocessing		Postprocessing							
	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Obj. value	Time (sec)	Obj. value	Lower bound	Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count
5×2	760	0.14	12	2090	0.13	4	55	0.09	1	1	0.41	65	1	2.89	37.80	124
10×2	2223	0.31	12	1.0e4	0.22	4	276	0.07	1	1	11.12	959	1	1.69	36.72	117
15×2	4408	0.30	8	2.4e4	0.16	4	1666	0.08	1	1	461.23	23976	1	2.99	35.30	108
20×2	38	0.55	52	4.3e4	0.20	4	5461	0.08	22	−6808	>1e4	221691	1	3.13	26.64	101
25×2	1.2e4	0.29	9	6.9e4	0.18	5	9216	0.04	28	−2.9e4	>1e4	74542	1	2.69	32.22	103
30×2	1.6e4	0.28	8	1.0e5	0.16	5	1.3e4	0.08	63	−6.3e4	>1e4	44355	1	2.91	22.09	100
35×2	38	0.34	24	1.4e5	0.20	5	1.8e4	0.05	129	−1.1e5	>1e4	30105	1	3.50	33.75	106
40×2	38	0.56	24	1.8e5	0.17	5	2.3e4	0.09	463	−1.7e5	>1e4	20657	1	4.80	36.52	105
45×2	38	0.72	30	2.3e5	0.25	5	3.0e4	0.06	547	−2.3e5	>1e4	14737	1	5.37	37.75	115
50×2	4.1e4	0.33	9	2.8e5	0.16	5	3.6e4	0.11	616	−3.1e5	>1e4	11341	1	5.04	42.28	103
100×2	38	0.36	32	1.2e6	0.17	5	1.5e5	0.28	1411	−1.7e6	>1e4	2049	1	4.61	30.25	113
250×2	9.3e5	0.28	8	7.2e6	0.25	5	9.0e5	1.23	3655	−1.2e7	>1e4	243	1	4.35	47.54	109
500×2	3.7e6	0.29	8	2.9e7	0.36	5	3.6e6	5.06	7873	−4.8e7	>1e4	46	1	8.19	80.26	138
1000×2	1.2e8	0.09	1	1.2e8	0.09	1	1.4e7	26.04	1.4e7	−1.9e8	>1e4	7	1	24.77	182.97	124

Proof For any feasible x, i.e., any x with exactly p ones and $(n-p)$ zeros, where $0 \le p \le \frac{n}{2}$,

$$\begin{aligned} f(x) &= -(n-1)p - \frac{\bar{p}}{n} + p(p-1)e^{(1)(1)} + (n(n-1) - p(p-1))e^{0} \\ &= (e-1)p^2 + (2-n-e)p - \frac{\bar{p}}{n} + n^2 - n \end{aligned}$$

for some $\bar{p}$ where $0 \le \bar{p} \le p$. By having the p ones occurring in the first $n/2$ variables of x, $f(x)$ assumes the minimum value of $(e-1)p^2 + (2-n-\frac{1}{n}-e)p + n^2 - n$ for a fixed value of p. Since this convex function in p is minimized when

$$p = -\frac{(2-n-\frac{1}{n}-e)}{2(e-1)}, \qquad p^* = \left[-\frac{(2-n-\frac{1}{n}-e)}{2(e-1)} \right]$$

will be an integer that minimizes this quadratic function over integral values of p lying in $[0, \frac{n}{2}]$. □

Both the GSA and the GAMS solvers were applied to this set of test problems with n ranging from 10 to 1000. We have 3 instances with $n \le 20$ ($n = 10, 16, 20$) to allow a better comparison of the performance and the computation time needed by the GSA and the GAMS solvers for solving the smaller test problems.

When applying GSA to (6.5), we can first convert the linear inequality constraint to an equality constraint by adding a non-negative slack variable s. An appropriate global smoothing function is then given by

$$\Phi(x, s) = -\sum_{j=1}^{n} \ln x_j - \sum_{j=1}^{n} \ln(1 - x_j) - \ln s, \tag{6.6}$$

so that we are handling the following relaxed equality-constrained problem:

$$\begin{aligned} &\text{Minimize} \quad f(x) + \mu \Phi(x, s) \\ &\text{subject to} \quad \sum_{i=1}^{n} x_i + s = \frac{n}{2}, \\ &\qquad\qquad 0 \le x \le e, \qquad s \ge 0. \end{aligned} \tag{6.7}$$

The initial μ parameter is set to 100 with a reduction ratio of 0.9. For $n \le 20$, a small initial γ parameter of 0.01 is sufficient for convergence to integral solutions. For larger dimensional problems, the initial γ parameter is progressively increased to reflect the increase in the objective as n increases. In particular, for $n = 50, 100, 200, 500$ and 1000, the initial γ parameter is set to 5, 20, 80, 250 and 500 respectively. The increment ratio for the γ parameter is set to the reciprocal of the μ-reduction ratio of 0.9, i.e., $\frac{10}{9}$ and the tolerance is set as $\epsilon_\mu = 10^{-3}$ for the termination criteria. We chose the initial iterate as $\frac{n}{2(n+1)}e$ based on uniform distribution of $\frac{n}{2}$ over the x and s variables. The objective values obtained by the GSA and the three GAMS solvers, as well as the respective computation time and number of

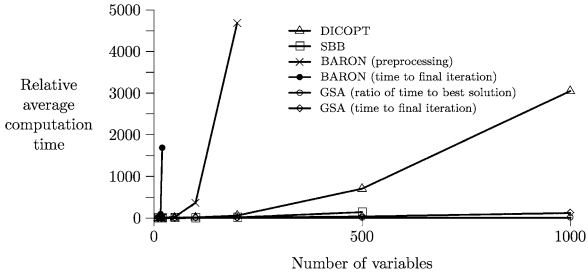

Fig. 7 Graph of relative average computation time required by the algorithms for solving instances of problem (6.5)

iterations required are given in Table 6. The global objective values are also included as a basis for performance comparisons.

From Table 6, we can see that GSA outperforms or performs as well as all the GAMS solvers in terms of the objective values obtained for all the 8 instances. In particular, GSA found the optimal solution to the test problems for 5 of the instances. For all the remaining instances, GSA has obtained solutions with less than 0.025% deviation from the optimal objective values. DICOPT was unable to obtain any optimal solution for all the instances and its solutions have percentage deviation from the optimal objective values ranging from 0.79% to 8.77%. SBB was able to obtain the optimal solution to 4 instances and for 3 of the remaining instances, SBB's solutions have percentage deviation from the optimal objective values ranging from 0.032% to 0.056%. However, SBB was unable to obtain any solution for the largest instance with 1000 variables. BARON was also able to obtain the optimal solution to 3 instances and for 3 of the remaining instances, BARON's solutions have percentage deviation from the optimal objective values ranging from 1.18% to 8.73%. However, BARON was unable to obtain any solution for the instances with 500 and 1000 variables. Despite running SBB and BARON on the Linux platform mentioned in Sect. 6.1, both solvers were still not able to produce any feasible solution for the respective large instances of the test set.

Figure 7 gives a plot of the relative average computation times required by GSA and the GAMS solvers for solving test problem (6.5). Some data points for BARON and SBB were not included for reasons discussed above. It can be seen that both GSA and SBB do not show a significant increase in the computation time needed with problem size, unlike BARON or DICOPT. In addition, GSA also appears to have a smaller rate of increase in computation time with problem size when compared with SBB.

7 Conclusions

We have described a global smoothing algorithm that enables optimization methods for continuous problems to be applied to certain types of discrete optimization problems. We were surprised by the quality of the solutions from what is a relatively unsophisticated implementation and without any need to fiddle with parameters. In the numerical experiments conducted, the performance of GSA relative to alternatives both in terms of the quality of solution and time to solution improved dramatically as

Table 6 Numerical comparison of algorithms for solving test problems (6.5) (minimization objective) with 10 to 1000 variables

Problem		DICOPT			SBB			BARON						GSA			
								Preprocessing		Postprocessing							
Size	Global obj. value	Best obj. value	Total time (sec)	Total itn. count.	Best obj. value	Total time (sec)	Total itn. count.	Obj. value	Time (sec)	Obj. value	Lower bound	Total time (sec)	Total itn. count	Best obj. value	Time to best (sec)	Total time (sec)	Total itn. count
10	73	79	0.4	30	73	34.7	1758	79	0.1	73	73	6.0	348	73	7.1	11.9	904
16	199	216	0.4	39	199	35.4	3985	216	0.2	199	199	498.0	14086	199	11.2	17.3	890
20	317	344	0.4	44	317	35.8	4203	344	0.2	317	315	>1e4	189748	317	12.1	16.7	883
50	2076	2256	5.1	84	2076	260.9	3593	2255	1.8	2100	1426	>1e4	481	2076	11.0	23.9	645
100	8424	9160	6.7	146	8429	108.9	3794	9159	22.7	9159	4950	>1e4	4	8425	19.0	50.7	575
200	33938	36911	25.1	262	33950	577.3	4385	35153	281.1	35153	19900	>1e4	1	33938	35.6	127.7	505
500	213022	231713	289.8	623	213090	5000.4	7107	–	>1e4	–	–	>1e4	–	213060	78.5	455.4	436
1000	853297	860074	1250.5	825	–	>1e4	–	–	>1e4	–	–	>1e4	–	853497	114.5	1460.2	395

the size of the problem increases, reflecting the encouraging feature of the low rate of growth in the work required by the new algorithm to solve problems as the size of the problem increases. Moreover, the memory requirements of the algorithm are also modest and typically add little to that required to prescribe the problem. Alternative methods are often memory intensive.

Given the difference in character to that of other algorithms, GSA could be used in conjunction with alternative methods by obtaining good solutions for the alternative methods, such as by providing a good upper bound in a branch-and-bound method. Certain improvement heuristics could also be applied to the solutions obtained by GSA in a hybrid algorithm framework. These are possible areas of future research, as well as the possibility of extending GSA to solve more general classes of nonlinear discrete optimization problems.

Acknowledgements The authors would like to thank the referees whose suggestions and comments led to improvements to the paper.

References

1. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific, Belmont (1995)
2. Boman, E.G.: Infeasibility and negative curvature in optimization. Ph.D. thesis, Scientific Computing and Computational Mathematics Program, Stanford University, Stanford (1999)
3. Borchardt, M.: An exact penalty approach for solving a class of minimization problems with boolean variables. Optimization **19**(6), 829–838 (1988)
4. Burer, S., Vandenbussche, D.: Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. Comput. Optim. Appl. (2007). doi: 10.1007/s10589-007-9137-6
5. Bussieck, M.R., Drud, A.S.: SBB: a new solver for mixed integer nonlinear programming, OR 2001 presentation. http://www.gams.com/presentations/or01/sbb.pdf (2001)
6. Cela, E.: The Quadratic Assignment Problem: Theory and Algorithms. Kluwer Academic, Dordrecht (1998)
7. Del Gatto, A.: A subspace method based on a differential equation approach to solve unconstrained optimization problems. Ph.D. thesis, Management Science and Engineering Department, Stanford University, Stanford (2000)
8. Dembo, R.S., Steihaug, T.: Truncated-Newton algorithms for large-scale unconstrained optimization. Math. Program. **26**, 190–212 (1983)
9. Du, D.-Z., Pardalos, P.M.: Global minimax approaches for solving discrete problems. In: Recent Advances in Optimization: Proceedings of the 8th French–German Conference on Optimization, Trier, 21–26 July 1996, pp. 34–48. Springer, Berlin (1997)
10. Duran, M.A., Grossmann, I.E.: An outer approximation algorithm for a class of mixed-integer nonlinear programs. Math. Program. **36**, 307–339 (1986)
11. Fiacco, A.V., McCormick, G.P.: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. Wiley, New York/Toronto (1968)
12. Forsgren, A., Gill, P.E., Murray, W.: Computing modified Newton directions using a partial Cholesky factorization. SIAM J. Sci. Comput. **16**, 139–150 (1995)
13. Forsgren, A., Murray, W.: Newton methods for large-scale linear equality-constrained minimization. SIAM J. Matrix Anal. Appl. **14**(2), 560–587 (1993)
14. Ge, R., Huang, C.: A continuous approach to nonlinear integer programming. Appl. Math. Comput. **34**, 39–60 (1989)
15. Gill, P.E., Murray, W.: Newton-type methods for unconstrained and linearly constrained optimization. Math. Program. **7**, 311–350 (1974)
16. Gill, P.E., Murray, W., Wright, M.: Practical Optimization. Academic Press, London (1981)
17. Golub, G.H., Van Loan, C.F.: Matrix Computation. John Hopkins University Press, Baltimore/London (1996)

18. Grossmann, I.E., Viswanathan, J., Vecchietti, A., Raman, R., Kalvelagen, E.: GAMS/DICOPT: a discrete continuous optimization package (2003)
19. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches. Springer, Berlin (1996)
20. Leyffer, S.: Deterministic methods for mixed integer nonlinear programming. Ph.D. thesis, Department of Mathematics & Computer Science, University of Dundee, Dundee (1993)
21. Lovász, L., Schrijver, A.: Cones of matrices and set-functions and 0-1 optimization. SIAM J. Optim. **1**, 166–190 (1991)
22. Mawengkang, H., Murtagh, B.A.: Solving nonlinear integer programs with large-scale optimization software. Ann. Oper. Res. **5**, 425–437 (1985)
23. Mitchell, J., Pardalos, P.M., Resende, M.G.C.: Interior point methods for combinatorial optimization. In: Handbook of Combinatorial Optimization, vol. 1, pp. 189–298 (1998)
24. Moré, J.J., Wu, Z.: Global continuation for distance geometry problems. SIAM J. Optim. **7**, 814–836 (1997)
25. Murray, W., Ng, K.-M.: Algorithms for global optimization and discrete problems based on methods for local optimization. In: Pardalos, P., Romeijn, E. (eds.) Heuristic Approaches. Handbook of Global Optimization, vol. 2, pp. 87–114. Kluwer Academic, Boston (2002), Chapter 3
26. Ng, K.-M.: A continuation approach for solving nonlinear optimization problems with discrete variables. Ph.D. thesis, Management Science and Engineering Department, Stanford University, Stanford (2002)
27. Pan, S., Tan, T., Jiang, Y.: A global continuation algorithm for solving binary quadratic programming problems. Comput. Optim. Appl. **41**(3), 349–362 (2008)
28. Pardalos, P.M.: Construction of test problems in quadratic bivalent programming. ACM Trans. Math. Softw. **17**(1), 74–87 (1991)
29. Pardalos, P.M.: Continuous approaches to discrete optimization problems. In: Di, G., Giannesi, F. (eds.) Nonlinear Optimization and Applications, pp. 313–328. Plenum, New York (1996)
30. Pardalos, P.M., Rosen, J.B.: Constrained Global Optimization: Algorithms and Applications. Springer, Berlin (1987)
31. Pardalos, P.M., Wolkowicz, H.: Topics in Semidefinite and Interior-Point Methods. Am. Math. Soc., Providence (1998)
32. Sahinidis, N.V.: BARON global optimization software user manual. http://archimedes.scs.uiuc.edu/baron/manuse.pdf (2000)
33. Tawarmalani, M., Sahinidis, N.V.: Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. Math. Program. **99**(3), 563–591 (2004)
34. Zhang, L.-S., Gao, F., Zhu, W.-X.: Nonlinear integer programming and global optimization. J. Comput. Math. **17**(2), 179–190 (1999)