Heuristic algorithms for the unconstrained binary quadratic programming problem

J.E. Beasley

December 1998

j.beasley@ic.ac.uk

http://mscmga.ms.ic.ac.uk/jeb/jeb.html

The Management School

Imperial College

London SW7 2AZ

England

ABSTRACT

In this paper we consider the unconstrained binary quadratic programming problem. This is the problem of maximising a quadratic objective by suitable choice of binary (zero-one) variables.

We present two heuristic algorithms based upon tabu search and simulated annealing for this problem. Computational results are presented for a number of publically available data sets involving up to 2500 variables.

An interesting feature of our results is that whilst for most problems tabu search dominates simulated annealing for the very largest problems we consider the converse is true.

This paper typifies a "multiple solution technique, single paper" approach, i.e. an approach that within the same paper presents results for a number of different heuristics applied to the same problem. Issues relating to algorithmic design for such papers are discussed.

Keywords: unconstrained binary (zero-one) quadratic programming; tabu search; simulated annealing

1. INTRODUCTION

The unconstrained binary quadratic programming problem, henceforth UBQP, the problem of maximising a quadratic objective by suitable choice of binary (zero-one) variables, can be formulated as follows. Let:

n be the size of the problem (number of variables)

$[q_{ij}]$ be a symmetric matrix of size n by n

x_i = 1 if variable i (i=1,...,n) is chosen

= 0 otherwise

then the problem is:

$$\text{maximise} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j \tag{1}$$

$$\text{subject to} \quad x_i \in [0,1] \quad i=1,\ldots,n \tag{2}$$

Equation (1) is a quadratic expression involving all pairs of variables, whilst equation (2) is the integrality constraint.

This problem is also sometimes referred to in the literature as the *unconstrained quadratic bivalent programming problem* [17]; the *unconstrained quadratic zero-one programming problem* [12]; the *quadratic zero-one programming problem* [22]; the *unconstrained pseudo-Boolean quadratic problem* [46]; the *unconstrained pseudo-Boolean quadratic zero-one programming problem* [45]; the *Boolean quadratic programming problem* [2] and the *binary quadratic program* [15].

Note here that it is well-known (e.g. see [37]) that any generalised objective involving both linear and quadratic terms such as:

$$\text{maximise} \quad \sum_{i=1}^{n} p_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \tag{3}$$

can be converted into the objective shown in equation (1) above involving the symmetric matrix $[q_{ij}]$ by noting that

$x_i=(x_i)^2$ (as $x_i\in[0,1]$) and setting $q_{ii}=p_i+a_{ii}$ $\forall i$; $q_{ij}=(a_{ij}+a_{ji})/2$ $\forall i\neq j$.

In the light of this UBQP, as defined above, can be regarded as a *general* representation of all unconstrained binary problems whose objective involves both linear and quadratic terms. Although a number of special cases of UBQP are solvable in polynomial time (see [3,34,41,42]), in general UBQP is NP-hard.

We would note here that the literature does not speak with one voice as to the definition of UBQP. By this we mean that:

(a) some authors regard it as a maximisation of a symmetric quadratic term, as in equation (1) above (e.g. [15])

(b) others regard it as the maximisation of the sum of a linear term and a quadratic term, as in equation (3) above (e.g. [30])

(c) others regard it as a minimisation of a symmetric quadratic term (e.g. [35])

(d) others regard it as the minimisation of the sum of a linear term and a quadratic term (e.g. [4]).

Although all of the above are (trivially) mathematical equivalent these different definitions can pose problems for the unwary reader.

Since the computational focus of this paper is a comparison with the recent work of Glover, Kochenberger and Alidaee [15] we have adopted here the same definition of UBQP as in that work.

UBQP, a binary quadratic program, can easily be

transformed into a binary linear program by linearising the quadratic term in a standard way. Defining variables y_{ij} to represent $x_i x_j$ we have that UBQP can be formulated as:

$$\text{maximise} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} y_{ij} \tag{4}$$

$$\text{subject to} \quad y_{ij} \leq x_i \quad i=1,\ldots,n \; j=1,\ldots,n \tag{5}$$

$$y_{ij} \leq x_j \quad i=1,\ldots,n \; j=1,\ldots,n \tag{6}$$

$$y_{ij} \geq x_i + x_j - 1 \quad i=1,\ldots,n \; j=1,\ldots,n \tag{7}$$

$$y_{ij} \in [0,1] \quad i=1,\ldots,n \; j=1,\ldots,n \tag{8}$$

$$x_i \in [0,1] \quad i=1,\ldots,n \tag{9}$$

Equations (5) and (6) ensure that y_{ij} must be zero if either of x_i or x_j are zero. Equation (7) ensures that y_{ij} is one if both x_i and x_j are one. Equations (8) and (9) are the integrality constraints.

The convex hull of solutions to equations (5)-(9) is known as the *Boolean quadric polytope* (also known as the *Boolean quadratic polytope* [28]). Recent work relating to characterising this polytope and defining facets can be found in [26,30,45-47].

The linear programming relaxation of the above program, equations (4)-(9), provides a bound on the optimal solution to UBQP and this is known as the roof dual bound [8,9,19].

UBQP has a number of applications. Phillips and Rosen [43] discuss how a molecular conformation problem can be formulated as an UBQP. Chardaire and Sutter [12] mention an application of the problem in cellular radio channel assignment.

It has long been known that UBQP is equivalent to the problem of finding a maximum cut in a graph (see [4,18]).

Pardalos and Rodgers [39] and Pardalos and Xue [40] have shown that a number of problems in graphs (maximum clique; maximum vertex packing; minimum vertex cover; maximum independent set; maximum weight independent set) can all be formulated as UBQP’s.

2. LITERATURE SURVEY

There have been a considerable number of papers presented in the literature relating to UBQP. In this section we review the most recent of these papers, structuring our review by considering exact algorithms and heuristic algorithms separately. References to work on the problem that occurred prior to the 1980's can be found both in the papers mentioned below and in Hansen [20].

2.1 Exact algorithms

Barahona, Jünger and Reinelt [4] presented an algorithm based on reducing the problem to a max-cut problem in a graph and using cutting planes derived for the max-cut problem. A branch and cut algorithm was used to solve the problem to optimality. Computational results were presented for problems of various sizes up to n=100.

Billionnet and Sutter [7] presented a tree search algorithm using a bound based upon the sum of three components, the first component using roof duality, the second using positive quadratic posiforms and the third using a posiform of degree four. Computational results were presented for problems of various sizes up to n=100.

Chardaire and Sutter [12] presented an algorithm for finding a bound on the problem by decomposing the original quadratic objective into a sum of pseudo-bilinear functions. The decomposition to use is found via (approximate) solution of the dual of a lagrangean decomposition of the initial problem. Computational results were presented for problems of

various sizes up to n=100.

Gulati, Gupta and Mittal [17] presented a tree search procedure based on enumerating local optima. Computational results were presented for problems of various sizes up to n=40.

Helmberg and Rendl [22] presented a tree search algorithm combining cutting planes and a bound based upon semidefinite relaxation. Computational results were presented for problems of various sizes up to n=450.

Kalantari and Bagchi [23] presented a tree search algorithm based on transforming the problem into an equivalent concave optimisation problem which can be tackled using an algorithm due to Kalantari and Rosen [24]. Computational results were presented for problems of various sizes up to n=50.

Palubeckis [32] presented a tree search algorithm incorporating transformation of subproblems based upon a heuristic solution. Computational results were presented for problems of various sizes up to n=247.

Pardalos and Rodgers [37] presented a tree search algorithm that uses a bound based upon the variables fixed at each node of the tree and consideration of the positive and negative elements of $[q_{ij}]$. Their algorithm incorporates a test to fix variables based upon a gradient expression. Computational results were presented for problems of various sizes up to n=200. They also presented FORTRAN code for the generation of standard test problems (see also Pardalos [33]).

Pardalos and Rodgers [38] presented a parallel branch and

bound algorithm implemented on a hypercube architecture. They used a bound based upon the sign of each q_{ij}. Computational results were presented for problems of various sizes up to n=100. See also [36].

2.2 Heuristic algorithms

Glover, Kochenberger and Alidaee [15] presented a heuristic based upon tabu search. In their heuristic a move corresponds to changing the value of a single variable (i.e. setting $x_i=1-x_i$ for some i). They refer to setting variables to one as a "constructive" phase and setting variables to zero as a "destructive" phase. They use a strategic oscillation scheme that alternates between constructive and destructive phases. Their approach employs recency and long-term frequency information related to "critical events", a critical event being a decrease in the objective function value. Computational results were presented for 45 test problems of various sizes up to n=500.

Lodi, Allemand and Liebling [27] presented a heuristic based upon genetic algorithms and scatter search. Their algorithm incorporates variable fixing (utilising the work of Pardalos and Rodgers [37]). As in Glover et al [15] they refer to setting variables to one as a "constructive" phase and setting variables to zero as a "destructive" phase and present a simple local search procedure that incorporates both phases. All children generated in their algorithm are subjected to this local search procedure in an attempt to improve them before they are entered into the population. Computational

results were presented for the same set of 45 test problems as were solved by Glover et al [15].

Palubeckis [31] presented two constructive heuristics with known worst-case performance. Computational results were presented for problems of various sizes up to n=306.

Pardalos and Jha [35] discussed the computational complexity of a number of problems related to UBQP and presented a simple local search heuristic for the problem. Computational results were presented for 20 test problems of size n=100.

3. HEURISTIC ALGORITHMS

In this section we outline the two heuristic algorithms based upon tabu search and simulated annealing that we have developed for solving UBQP. We also comment on the algorithmic design of our heuristics and on applying a genetic algorithm to the problem.

3.1 Tabu search

Tabu search (TS) is a local search heuristic due to Glover [14] and Hansen [21]. In TS the fundamental concept is that of a "move", a systematic operator that, given a single starting solution, generates a number of other possible solutions. In local search terms these other solutions are the "neighbourhood" of the single starting solution. Note here that these solutions may, or may not, be feasible. From the neighbourhood the "best" solution is chosen to become the new starting solution for the next iteration and the process repeats. This "best" solution may either be the first improving solution encountered as the move operator enumerates the neighbourhood, or it may be based upon complete enumeration of the neighbourhood.

In order to prevent cycling a list of "tabu moves" is employed. Typically this list prohibits certain moves which would lead to the revisiting of a previously encountered solution. This list of tabu moves is updated as the algorithm proceeds so that a move just added to the tabu list is removed from the tabu list after a certain number of iterations (the "tabu tenure") have passed. Tabu moves are sometimes allowed

to be made however if they lead to an improved feasible solution (an "aspiration criteria"). A more comprehensive overview of TS can be found in [1,16,44].

3.2 Tabu search heuristic

In our TS heuristic a move corresponds to changing the value of a single variable (i.e. setting $x_i=1-x_i$ for some i). This is the natural move for UBQP and was used by Glover et al [15]. We started from the all-zero solution ($x_i=0$ i=1,...,n) and, at each solution, examined all non-tabu moves. If a move was found that led to an improved solution then it was taken immediately, otherwise from the set of all non-tabu moves the move that led to the best solution was taken.

Limited computational experience indicated that it was effective to apply a simple local search procedure each time an improved solution was found. This procedure attempts to improve that solution further by examining all possible moves (irrespective of their tabu status) and making any moves that improve the solution. Our local search procedure is shown in pseudocode in Algorithm 1 and our complete TS heuristic is shown in pseudocode in Algorithm 2.

The (arbitrary) termination criterion in our TS heuristic shown in Algorithm 2 was that we could only examine max(500000,5000n) solutions. This is a key computational point. By far the most computationally expensive part of our heuristic is evaluating a solution (i.e. computing the objective function value as given by equation (1)). A naive implementation of this evaluation would require $O(n^2)$

operations. However in our TS heuristic we are evaluating a new solution which differs by just a single move (i.e. by only one variable value) from an old solution. As the objective function value of this old solution is known the objective function evaluation of the new solution can be done by focusing on the variable that has changed in value. In this way evaluating a new solution can be accomplished in O(n) operations.

3.3 Simulated annealing

Simulated annealing (SA) originated in an algorithm to simulate the cooling of material in a heat bath [29] but its use for optimisation problems originated with Kirkpatrick, Gelatt and Vecchi [25] and Cerny [10].

SA has much in common with TS in that they both examine potential moves from a single starting solution. SA incorporates a statistical component in that moves to worse solutions are accepted with a specified probability that decreases over the course of the algorithm.

This probability is related to what is known as the "temperature". More precisely, a move that worsens the objective function value by Δ is accepted with a probability proportional to $e^{-\Delta/T}$, where T is the current temperature. The higher the temperature T, the higher the probability of accepting the move. Hence this probability decreases as the temperature decreases.

In SA the temperature is reduced over the course of the algorithm according to a "cooling schedule" which specifies

the initial temperature and the rate at which temperature decreases. A common cooling schedule is to reduce the temperature T by a constant factor α ($0<\alpha<1$) using $T=\alpha T$ at regular intervals. A more comprehensive overview of SA can be found in [1,44].

3.4 Simulated annealing heuristic

Our SA heuristic is similar to our TS heuristic and can be seen in pseudocode in Algorithm 3. The initial temperature was set equal to the problem size n and α was set equal to 0.995. In contrast to our TS heuristic limited computational experience indicated that it was ineffective to apply a simple local search procedure each time an improved solution was found. Rather that local search procedure was applied once only at the end of the SA heuristic.

Note here that, as far as we are aware, this paper represents the first reported application of simulated annealing to UBQP.

3.5 Algorithmic design

There is an important point relating to the algorithmic design of the heuristics presented above, and to their subsequent computational comparison presented below, that we wish to emphasise here.

The majority of papers in the literature that deal with heuristics (not just for UBQP but also for many other combinatorial optimisation problems) apply just a single solution technique (e.g. tabu search) to the problem under

consideration. A computational comparison is then carried out with the algorithms of other workers, increasingly in recent years on publically available test problems. As such it is clearly legitimate (and indeed the author freely admits to having done this himself) to build into an algorithm what might colloquially be phrased "bells and whistles", algorithmic tricks/steps (***extra intellectual effort***) that computationally, mean that the algorithm generates better results than it otherwise would.

Recently however the author (e.g. as in [11]) has tried to move beyond this "single solution technique, single paper" approach to a "multiple solution technique, single paper" approach, i.e. to an approach that within the same paper presents results for a number of different heuristics applied to the same problem. As such the key algorithmic design point is whether, or not, it is legitimate to build "bells and whistles" into one algorithm at the expense of another.

Consider the two heuristics presented above. We could have:

(a) built many extra tricks/steps (invested much more intellectual effort) in the tabu search algorithm in an attempt to improve its computational performance

(b) not built any such tricks/steps (not invested as much intellectual effort) in the simulated annealing algorithm.

As such we may have obtained different computational results.

However, ***we strongly believe***, that this is not a legitimate approach to algorithm comparison within a "multiple

solution technique, single paper" approach. Simply put, it is not appropriate to attempt to bias the results by investing more intellectual effort in one algorithm than in another.

Whilst, of necessity, it must remain an imprecise calculation as to the amount of intellectual effort invested in an algorithm we have tried, in designing the heuristic algorithms presented above, to bear this point in mind and not to bias our results accordingly.

On a personal note we would comment that whilst this "multiple solution technique, single paper" approach does nothing to length one's publication list (indeed quite the opposite) we do believe that more work of this type should be carried out.

3.6 Genetic algorithm

We did experiment with a genetic algorithm (GA) for UBQP. The results are not reported here because we came to believe that, as the problem size increases, a population heuristic such as a GA (which explicitly works with a population of solutions) could never compete computationally with a single solution heuristic such as TS or SA which make a succession of small changes to a single solution.

Our reasoning for this is based on the point discussed above concerning the computational cost of evaluating a solution. As noted above, within TS or SA a new solution can be evaluated in O(n) operations. However in a GA the child can be distinctly different from the parents. This would potentially entail $O(n^2)$ operations to compute the objective

function value of the child. Although it is clear that (say) if the child only differed from one (or other) of the parents with regard to K variable values this evaluation might be accomplished in O(Kn) operations the essential point remains; namely that solution evaluation in a GA for UBQP can be computationally much more expensive than in a TS or a SA algorithm for UBQP.

Consider the largest problems, with n=2500, dealt with in this paper. Simply evaluating one GA child could (in the worst case) take as long as evaluating 2500 TS or SA solutions. Even initialising a GA with a population of (say) 100 randomly generated solutions would be equivalent to evaluating 250,000 TS or SA solutions. These figures indicate the substantial computational disadvantage under which a GA for UBQP must labour in comparison to TS or SA.

Given this reasoning our current view is that the only way to successfully apply a GA to UBQP is to:

(a) only consider relatively small problems; and/or

(b) invest "intellectual effort" in the algorithm (see the algorithmic design discussion above).

However, as will become apparent below, our TS and SA algorithms are (in our view) successful without this extra intellectual effort and so, in the light of the algorithmic design issues outlined before for a "multiple solution technique, single problem" paper such as the one presented here, we have not pursued the GA approach.

We would comment here that we believe that the Lodi et al [27] algorithm, which is a population heuristic, relies on

"intellectual effort" to limit the number of child solutions that have to be evaluated.

4. COMPUTATIONAL RESULTS

In this section we present computational results for the two heuristic algorithms (TS and SA) we have presented above for solving UBQP. Since we will be comparing these algorithms with the heuristics of Glover, Kochenberger and Alidaee [15] and of Lodi, Allemand and Liebling [27] we will refer to their heuristics as GKA and LAL respectively.

4.1 Glover et al [15] test problems

To test our heuristics we first solved the 45 test problems considered in Glover et al [15]. These problems have various sizes (up to n=500) and varying characteristics with respect to the $[q_{ij}]$ matrix.

The results for these test problems are shown in Table 1. In that table we show, for each problem, the results reported in [15] for GKA with respect to: the best solution value; the time taken to first find this solution; and the total execution time. All computer times are in seconds on a Pentium 90 pc.

For the first 25 problems shown in Table 1 (problems 1a-8a, 1b-10b and 1c-7c) the solution values given are known to be optimal. For the remaining problems the optimal values are not known.

Also in Table 1 we show, both for LAL and for the tabu search and simulated annealing heuristics reported in this paper:

(a) the difference between the best solution found by GKA and the best solution found by those heuristics (i.e. GKA

solution value - heuristic solution value)

(b) the time taken to first find this best solution (in seconds)

(c) the total execution time (in seconds).

The computation times for LAL relate to a Silicon Graphics INDY R10000sc (195Mhz) and for our work relate to a Silicon Graphics Indigo (R4000, 100MHz, 48MB main memory).

We would mention here that both GKA and LAL are run with different parameter values for different sized problems. Our TS and SA algorithms, by contrast, use a consistent set of parameter values for all problem sizes.

With regard to quality of results:

(a) it is clear that GKA and LAL are equivalent, producing the same solution in all 45 test problems

(b) our TS algorithm is only slightly less effective, producing the same solution in 42 of the 45 problems. However the deviations from the best known solution values are extremely small (an average deviation from the best known solution of just 0.00035%)

(c) our SA algorithm is the least competitive of these heuristics, but even it finds the best known solution in 38 of the 45 problems with an average deviation from the best known solution of just 0.050%.

A computational comparison of the results shown in Table 1 is difficult as different computers have been used, as well as different stopping criteria. From Dongarra [13] we can make an approximate comparison of the results shown in Table 1 by using scaling values for the achievable speeds of these

computers as 15 for our Silicon Graphics Indigo and 11 for a Pentium 90 pc.

The current (late 1998) version of Dongarra [13] does not contain a value for a Silicon Graphics INDY R10000sc. Lodi et al [27] state that they believe that this machine is equivalent to a 200 Mhz pc. Utilising this statement, together with technical information from the Silicon Graphics web site (*http://www.sgi.com/*) and the SPEC benchmark web site (*http://www.spec.org/*) our best estimate for the achievable speed of their machine is 50.

Using these scaling values and the average time to first find the best solution shown in Table 1 the ratio GKA:LAL:TS:SA is given by 181(11):5(50):63(15):15(15) = 1991:250:945:225. We would therefore make the following observations:

(a) our TS heuristic finds its best solution in approximately half of the GKA time (i.e. it converges approximately twice as fast)

(b) our SA heuristic converges approximately four times faster than our TS heuristic (and at least eight times faster than GKA)

(c) faster convergence is of interest as it implies more opportunity for repeated applications of the algorithm to the same problem (indeed LAL, for example, utilises just such a restart strategy)

(d) restarting our TS heuristic just once (with a random starting solution instead of the all-zero starting solution) meant that the number of cases in which TS

yielded a worse solution than GKA dropped from the three shown in Table 1 to just one

(e) restarting our SA heuristic just once (with a different random starting solution) meant that the number of cases in which SA yielded a worse solution than GKA dropped from the seven shown in Table 1 to three.

Overall though our conclusion must be that LAL is the dominant heuristic of the four heuristics shown in Table 1. GKA and TS vie for second place (depending upon the emphasis placed on quality of results or time taken) whilst SA trails the other heuristics. Were we to allow multiple restarts then both TS and SA (because of their faster convergence) would become much more competitive with GKA.

However we would like to return here to the algorithmic design issue mentioned previously. It is clear (at least to us) that both GKA and LAL contain much more intellectual effort ("bells and whistles") than are contained in our heuristics.

One might expect therefore both GKA and LAL to be ***far superior*** to our simple heuristics. That this is not the case is, we believe, a validation of the "multiple solution technique, single paper" approach we have followed in this paper. Following such an approach forces one to balance intellectual effort and focus on key algorithmic design features, rather than allowing one to incorporate any number of algorithmic tricks/steps such as is commonly done in the "single solution technique, single paper" approach.

4.2 Other test problems

In order to compare our heuristics more fully we also randomly generated a larger series of 60 test problems (n=50/100/250/500/1000/2500; 10 problems for each value of n) with constant characteristics (10% density; all q_{ij} entries being integers uniformly drawn from [-100,+100]). These problems are in contrast to the problems considered in Table 1 which have varying characteristics.

As far as we are aware problems of this size are significantly larger than problems considered to date in the literature.

Note here that the test problems solved in this paper are publically available from OR-Library [5,6], email the message *bqpinfo* to *o.rlibrary@ic.ac.uk* or see *http://mscmga.ms.ic.ac.uk/jeb/orlib/bqpinfo.html*. By making our problems publically available we hope to stimulate work on heuristic techniques for UBQP.

The results for these test problems are shown in Table 2. In that table we show the best known solution value (either from TS or SA). In addition we show (both for TS and for SA):

(a) the difference between the best known solution value and the best solution found by each of our heuristics (i.e. best known solution value - heuristic solution value)

(b) the time taken to first find this best solution (in seconds)

(c) the total execution time (in seconds).

As before these times relate to a Silicon Graphics Indigo (R4000, 100MHz, 48MB main memory).

It is interesting to note from Table 2 that for $n \leq 1000$ the comparative performance of TS and SA is as in Table 1 (TS getting better quality results slower).

However for n=2500 SA outperforms TS. For the ten problems with n=2500 TS finds the best known solution 4 times, has an average percentage deviation from the best known solution of 0.045% and an average time to solution of 35071. SA dominates TS with respect to all of these measures: finding the best known solution 6 times, having an average percentage deviation from the best known solution of 0.015% and an average time to solution of 11415.

Whether the LAL heuristic of Lodi et al [27] (a population based approach), which was the dominant heuristic on the problems shown in Table 1, would remain dominant when applied to the much larger problems considered in Table 2 is currently an open question. As our results for TS and SA show, relative performance can change as problem size increases.

However we would repeat here our comments made previously that our experience for UBQP has been that, as problem size increases, getting a population heuristic to compete computationally with single solution heuristics (such as tabu search or simulated annealing) is difficult since solution evaluation in a population heuristic for UBQP can be computationally much more expensive than in a tabu search or simulated annealing algorithm for UBQP.

5. CONCLUSIONS

In this paper we have considered the unconstrained binary quadratic programming problem and have presented two heuristic algorithms based upon tabu search and simulated annealing. Computational results were given for publically available test problems that are significantly larger than those solved by other workers.

For most problems tabu search dominated simulated annealing but for the very largest problems considered the converse was true.

This paper was an example of the "multiple solution technique, single paper" approach, and issues relating to algorithmic design for such papers were discussed.

REFERENCES

[1] E.H.L. Aarts and J.K. Lenstra (editors), *Local search in combinatorial optimization*. Wiley, (1997).

[2] K.M. Anstreicher, On the equivalence of convex programming bounds for boolean quadratic programming. Working paper (1998) available from the author at Department of Management Sciences, University of Iowa, Iowa City, IA 52242, USA.

[3] F. Barahona, A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics* 13 (1986) 23-26.

[4] F. Barahona, M. Jünger and G. Reinelt, Experiments in quadratic 0-1 programming. *Mathematical Programming* 44 (1989) 127-137.

[5] J.E. Beasley, OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41 (1990) 1069-1072.

[6] J.E. Beasley, Obtaining test problems via Internet. *Journal of Global Optimization* 8 (1996) 429-433.

[7] A. Billionnet and A. Sutter, Minimization of a quadratic pseudo-Boolean function. *European Journal of Operational Research* 78 (1994) 106-115.

[8] E. Boros, Y. Crama and P.L. Hammer, Upper-bounds for quadratic 0-1 maximization. *Operations Research Letters* 9 (1990) 73-79.

[9] E. Boros, Y. Crama and P.L. Hammer, Chvatal cuts and odd cycle inequalities in quadratic 0-1 optimization. *SIAM Journal of Discrete Mathematics* 5 (1992) 163-177.

[10] V. Cerny, Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45 (1985) 41-51.

[11] T.-J. Chang, N. Meade, J.E. Beasley and Y.M. Sharaiha, Heuristics for cardinality constrained portfolio optimisation. Working paper (1998) available from the third author at The Management School, Imperial College, London SW7 2AZ.

[12] P. Chardaire and A. Sutter, A decomposition method for quadratic zero-one programming. *Management Science* 41 (1995) 704-712.

[13] J.J. Dongarra, Performance of various computers using standard linear equations software. Working paper (1998) available from the author at Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, USA. Also available at

http://www.netlib.org/benchmark/performance.ps

[14] F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* 13 (1986) 533-549.

[15] F. Glover, G.A. Kochenberger and B. Alidaee, Adaptive memory tabu search for binary quadratic programs. *Management Science* 44 (1998) 336-345.

[16] F.W. Glover and M. Laguna, *Tabu search*. Kluwer Academic Publishers, (1997).

[17] V.P. Gulati, S.K. Gupta and A.K. Mittal, Unconstrained quadratic bivalent programming problem. *European Journal of Operational Research* 15 (1984) 121-125.

[18] P.L. Hammer, Some network flow problems solved with pseudo-Boolean programming. *Operations Research* 13 (1965) 388-399.

[19] P.L. Hammer, P. Hansen and B. Simeone, Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming* 28 (1984) 121-155.

[20] P. Hansen, Methods of nonlinear 0-1 programming. *Annals of Discrete Mathematics* 5 (1979) 53-70.

[21] P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming. Presented at the *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).

[22] C. Helmberg and F. Rendl, Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Mathematical Programming* 82 (1998) 291-315.

[23] B. Kalantari and A. Bagchi, An algorithm for quadratic zero-one programs. *Naval Research Logistics* 37 (1990) 527-538.

[24] B. Kalantari and J.B. Rosen, An algorithm for global minimization of linearly constrained concave quadratic functions. *Mathematics of Operations Research* 12 (1987) 544-561.

[25] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing. *Science* 220 (1983) 671-680.

[26] C.-W. Ko, J. Lee and E. Steingrimsson, The volume of relaxed Boolean-quadric and cut polytopes. *Discrete Mathematics* 163 (1997) 293-298.

[27] A. Lodi, K. Allemand and T.M. Liebling, An evolutionary heuristic for quadratic 0-1 programming. Working paper (1998) available from the second author at Departement de Mathematiques, Ecole Polytechnique Federale de Lausanne, CH 1015 Lausanne, Switzerland.

[28] A. Mehrotra, Cardinality constrained Boolean quadratic polytope. *Discrete Applied Mathematics* 79 (1997) 137-154.

[29] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, Equation of state calculations by fast computing machines. *Journal of Chemical Physics* 21 (1953) 1087-1092.

[30] M. Padberg, The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming* 45 (1989) 139-172.

[31] G. Palubeckis, Heuristics with a worst-case bound for unconstrained quadratic 0-1 programming. *Informatica* 3 (1992) 225-240.

[32] G. Palubeckis, A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing* 54 (1995) 283-301.

[33] P.M. Pardalos, Construction of test problems in quadratic bivalent programming. *ACM Transactions on Mathematical Software* 17 (1991) 74-87.

[34] P.M. Pardalos and S. Jha, Graph separation techniques for quadratic zero-one programming. *Computers and Mathematics with Applications* 21 (1991) 107-113.

[35] P.M. Pardalos and S. Jha, Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters* 11 (1992) 119-123.

[36] P.M. Pardalos and G.P. Rodgers, Parallel branch and bound algorithms for unconstrained quadratic zero-one programming. In R. Sharda et al (editors) *Impacts of recent computer advances on Operations Research* (1989) 131-143, North-Holland, Amsterdam.

[37] P.M. Pardalos and G.P. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing* 45 (1990) 131-144.

[38] P.M. Pardalos and G.P. Rodgers, Parallel branch and bound algorithms for quadratic zero-one programs on the hypercube architecture. *Annals of Operations Research* 22 (1990) 271-292.

[39] P.M. Pardalos and G.P. Rodgers, A branch and bound algorithm for the maximum clique problem. *Computers & Operations Research* 19 (1992) 363-375.

[40] P.M. Pardalos and J. Xue, The maximum clique problem. *Journal of Global Optimization* 4 (1994) 301-328.

[41] J.C. Picard and H.D. Ratliff, A graph-theoretic equivalence for integer programs. *Operations Research* 21 (1973) 261-269.

[42] J.C. Picard and H.D. Ratliff, Minimum cuts and related problems. *Networks* 5 (1975) 357-370.

[43] A.T. Phillips and J.B. Rosen, A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization* 4 (1994) 229-241.

[44] C.R. Reeves (editor), *Modern heuristic techniques for combinatorial problems*. Blackwell Scientific Publications, Oxford, (1993).

[45] H.D. Sherali, Y. Lee and W.P. Adams, A simultaneous lifting strategy for identifying new classes of facets for the Boolean quadric polytope. *Operations Research Letters* 17 (1995) 19-26.

[46] C. De Simone, The cut polytope and the Boolean quadric polytope. *Discrete Mathematics* 79 (1989) 71-75.

[47] C. De Simone, A note on the Boolean quadric polytope. *Operations Research Letters* 19 (1996) 115-116.

X_i is the value for variable i in the current solution
V^* is the best solution value found so far
t is the iteration counter

begin local_search($[X_i]$,V^*,t)
improved:=false /* set flag that marks whether an improved solution found or not */
for k=1 to n **do** /* examine all variables */
t:=t+1 /* increment iteration counter */
X_k:=1-X_k /* make the move for variable k */

V:= $\sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} X_i X_j$ /* evaluate new solution */

if V > V^* **then** /* check for improved solution */
V^*:=V /* record improved solution */
improved:=true /* set improved flag */
else
X_k:=1-X_k /* reset variable k */
end if
end for
if improved **then** repeat this procedure **else return**
end

Algorithm 1: Local search

X_i is the value for variable i in the current solution
V^* is the best solution value found so far
V^{**} is the best solution value associated with a neighbour of $[X_i]$
L_i is the tabu value for variable i, where $L_i=0$ if the variable is not tabu
L^* is the tabu tenure value
t is the iteration counter

begin
$X_i:=0$ i=1,...,n /* set the starting solution */
$V^*:=0$ /* initialise best solution value */
$L_i:=0$ i=1,...,n /* initialise tabu values */
$L^*:=\min(20,n/4)$ /* set tabu tenure */
$T^*:=\max(500000,5000n)$ /* set maximum number of iterations */
t:=0 /* initialise iteration counter */
while $t<T^*$ **do** /* T^* iterations in all */
$V^{**}:=-\infty$ /* initialise best neighbour value */
for k=1 to n and $L_k=0$ **do** /* examine all non-tabu variables */
t:=t+1 /* increment iteration counter */
$X_k:=1-X_k$ /* make the move for variable k */

$V:=\sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}X_iX_j$ /* evaluate new solution */

if $V > V^*$ **then** /* check for improved solution */
K:=k /* record variable associated with the move */
local_search($[X_i]$,V^*,t) /* apply the local search procedure */
go to done:
end if

$X_k:=1-X_k$ /* reset variable k */

if $V > V^{**}$ **then** /* check for improved neighbouring solution */
K:=k /* record variable associated with the move */
$V^{**}:=V$ /* record solution value */
end if
end for

$X_K:=1-X_K$ /* make the move for the chosen variable K */

done: $L_i:=\max(0,L_i-1)$ i=1,...,n /* reduce all tabu values by one */
$L_K:=L^*$ /* tabu the chosen variable */

end while
return V^*
end

Algorithm 2: TS heuristic

X_i is the value for variable i in the current solution
V^* is the best solution value found so far
Y_i is the value of variable i in V^*
V^{**} is the solution value associated with the current solution
t is the iteration counter

begin
initialise X_i i=1,...,n randomly /* randomise the starting solution */
Y_i:=X_i i=1,...,n /* set best solution */

$V^*:= \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}X_iX_j$ /* initialise best solution value */
$V^{**}:=V^*$ /* set current solution value */
T:=n /* initialise SA parameters */
α:=0.995
T^*:=max(500000,5000n) /* set maximum number of iterations */
t:=0 /* initialise iteration counter */
while t<T^* **do** /* T^* iterations in all */
t:=t+1 /* increment iteration counter */
randomly select a variable k
X_k:=1-X_k /* make the move for variable k */

$V:= \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}X_iX_j$ /* evaluate new solution */

if V > V^* **then** /* check for improved solution */
V^*:=V /* record the improved solution */
Y_i:=X_i i=1,...,n
end if

if V > V^{**} **then** /* check if new solution better than current solution */
V^{**}:=V /* change V^{**} */
else
r:=a random number drawn from [0,1]
if r < exp[-|V^{**}-V|/T] **then** /* check for accept worst solution */
V^{**}:=V /* change V^{**} */
else
X_k:=1-X_k /* reset variable k */
end if
end if

T:=αT /* reduce temperature */

end while
local_search([Y_i],V^*,t) /* apply the local search procedure */
return V^*
end

Algorithm 3: SA heuristic

Problem number [15]	n	GKA [15]			LAL [27]			TS			SA		
		Solution value	Time to solution	Total time	Solution difference	Time to solution	Total time	Solution difference	Time to solution	Total time	Solution difference	Time to solution	Total time
1a	50	3414	< 1	37	-	< 1	1	-	< 1	16	-	< 1	23
2a	60	6063	< 1	60	-	< 1	1	-	< 1	22	-	< 1	30
3a	70	6037	15	71	-	< 1	1	-	7	21	-	< 1	26
4a	80	8598	14	93	-	< 1	1	-	10	26	-	1	31
5a	50	5737	7	38	-	< 1	1	-	< 1	19	-	< 1	24
6a	30	3980	< 1	15	-	< 1	1	-	< 1	11	-	< 1	17
7a	30	4541	< 1	14	-	< 1	1	-	< 1	12	-	< 1	18
8a	100	11109	17	141	-	< 1	1	-	< 1	29	-	1	40
1b	20	133	< 1	2	-	< 1	1	-	< 1	6	-	< 1	9
2b	30	121	< 1	4	-	< 1	1	-	< 1	7	-	< 1	10
3b	40	118	< 1	5	-	< 1	1	-	< 1	9	-	< 1	11
4b	50	129	< 1	8	-	< 1	1	-	< 1	10	-	< 1	13
5b	60	150	< 1	11	-	< 1	1	-	< 1	11	-	< 1	12
6b	70	146	< 1	14	-	< 1	1	-	2	12	-	< 1	12
7b	80	160	< 1	18	-	< 1	1	-	< 1	13	-	< 1	13
8b	90	145	5	21	-	< 1	1	-	11	15	-	< 1	14
9b	100	137	11	27	-	< 1	1	-	4	15	2	1	15
10b	125	154	5	46	-	< 1	1	-	2	21	-	2	22
1c	40	5058	< 1	27	-	< 1	1	-	< 1	17	-	< 1	20
2c	50	6213	< 1	40	-	< 1	1	-	< 1	22	-	< 1	30
3c	60	6665	3	64	-	< 1	1	-	< 1	23	-	< 1	31
4c	70	7398	10	74	-	< 1	1	-	< 1	24	-	< 1	31
5c	80	7362	41	100	-	< 1	1	-	< 1	27	-	2	37
6c	90	5824	49	111	-	< 1	1	-	6	28	-	2	35
7c	100	7225	7	140	-	< 1	1	-	6	32	-	2	40
1d	100	6333	3	60	-	< 1	1	-	< 1	32	-	2	41
2d	100	6579	18	56	-	< 1	1	-	< 1	29	-	2	37
3d	100	9261	3	62	-	< 1	1	-	< 1	31	-	2	40
4d	100	10727	7	59	-	< 1	1	-	< 1	31	-	2	38
5d	100	11626	53	64	-	< 1	1	-	12	32	-	1	41
6d	100	14207	32	64	-	< 1	1	-	2	32	50	1	37
7d	100	14476	23	63	-	< 1	1	-	< 1	36	-	1	43
8d	100	16352	11	63	-	< 1	1	-	< 1	36	-	2	45
9d	100	15656	1	58	-	< 1	1	-	< 1	33	-	1	38
10d	100	19102	2	70	-	< 1	1	-	< 1	39	-	< 1	48
1e	200	16464	374	876	-	2	5	-	26	128	6	14	141
2e	200	23395	149	913	-	< 1	5	-	63	130	-	19	147
3e	200	25243	56	1069	-	< 1	5	-	3	130	-	10	149
4e	200	35594	21	1054	-	< 1	5	-	< 1	149	-	15	168
5e	200	35154	284	1007	-	< 1	5	-	18	149	-	15	168
1f	500	61194	2530	4925	-	43	60	-	53	960	11	116	965
2f	500	100161	1055	5005	-	43	60	3	619	1031	3	130	1033
3f	500	138035	205	4900	-	59	60	14	819	1076	-	127	1077
4f	500	172771	2520	4835	-	9	60	-	1107	1156	621	89	1142
5f	500	190507	590	5090	-	59	60	5	62	1182	9	115	1210
Average			181	700		5	8		63	150		15	159

Table 1: Results for the Glover et al [15] problems

Notes:
(a) for problems 1f-5f the times quoted above have been adjusted (using the information given in [15]) to account for use of a different computer
(b) for times given as < 1 a figure of 0.5 has been assumed in computing the average values given above

n	Problem number	Best known solution value	TS			SA		
			Solution difference	Time to solution	Total time	Solution difference	Time to solution	Total time
50	1	2098	-	< 1	14	-	< 1	19
	2	3702	-	< 1	16	-	< 1	20
	3	4626	-	< 1	17	-	< 1	21
	4	3544	-	< 1	16	-	< 1	21
	5	4012	-	< 1	16	-	< 1	20
	6	3693	-	< 1	16	-	< 1	22
	7	4520	-	< 1	17	-	< 1	22
	8	4216	-	< 1	17	-	< 1	22
	9	3780	-	< 1	17	-	< 1	22
	10	3507	-	< 1	17	-	< 1	21
100	1	7970	-	5	34	28	1	31
	2	11036	-	8	35	-	2	34
	3	12723	-	8	37	-	2	34
	4	10368	-	< 1	33	-	1	33
	5	9083	-	< 1	36	-	2	33
	6	10210	-	5	36	-	2	34
	7	10125	-	24	36	-	3	32
	8	11435	-	2	36	-	2	31
	9	11455	-	< 1	35	-	1	32
	10	12565	-	36	38	-	2	36
250	1	45607	-	45	238	-	27	226
	2	44810	-	16	239	-	17	226
	3	49037	-	49	254	-	19	240
	4	41274	-	13	234	-	20	218
	5	47961	-	8	245	-	22	232
	6	41014	-	25	240	-	16	221
	7	46757	-	43	250	-	19	232
	8	35726	-	17	225	-	23	212
	9	48916	-	133	246	-	14	229
	10	40442	-	3	235	-	20	218
500	1	116586	-	681	956	-	113	1006
	2	128223	-	326	979	19	112	1009
	3	130812	-	438	987	-	104	1030
	4	130097	-	477	1003	20	142	1061
	5	125487	-	499	964	172	101	1030
	6	121719	-	69	966	-	111	1028
	7	122201	-	518	952	-	120	1014
	8	123559	-	245	1006	90	135	1050
	9	120798	1	85	954	-	125	998
	10	130619	-	164	971	-	112	1012
1000	1	371438	-	432	4608	304	1083	7150
	2	354932	-	2274	4514	295	983	6794
	3	371226	153	2829	4518	-	841	6943
	4	370560	-	1935	4580	295	1119	7011
	5	352736	-	1186	4512	439	975	6939
	6	359452	-	3492	4444	139	1001	6749
	7	370999	-	922	4546	184	965	6885
	8	351836	-	243	4461	835	1079	6961
	9	348732	-	2478	4488	423	891	6626
	10	351415	7	3553	4474	-	923	6734
2500	1	1515011	40	51461	52011	-	11896	63080
	2	1468850	156	25440	51659	-	11588	62787
	3	1413083	2362	31186	49101	-	10777	60963
	4	1506943	701	19140	50642	-	11263	63018
	5	1491796	-	34293	51194	331	13189	63470
	6	1468427	727	42966	51669	-	10962	63310
	7	1478654	2595	48243	50798	-	11322	62833
	8	1484199	-	12721	49861	1246	10196	61918
	9	1482306	-	41408	51873	472	10852	62978
	10	1482354	-	43855	50981	188	12108	62777

Table 2: Results for other problems