Efficient evaluations for solving large 0–1 unconstrained quadratic optimisation problems

Fred Glover*

OprTek Systems, Inc.,
1919 Seventh Street,
Boulder, CO 80302, USA
E-mail: glover@opttek.com
*Corresponding author

Jin-Kao Hao

Laboratoire d'Etude et de Recherche en Informatique (LERIA),
Université d'Angers,
2 Boulevard Lavoisier,
49045 Angers Cedex 01, France
E-mail: jin-kao.hao@univ-angers.fr

Abstract: We provide a method for efficiently evaluating moves that complement values of 0–1 variables in search methods for binary unconstrained quadratic optimisation problems. Our method exploits a compact matrix representation and offers further improvements in speed by exploiting sparse matrices that arise in large-scale applications. The resulting approach, which works with integer or real data, can be applied to improve the efficiency of a variety of different search processes, especially in the case of commonly encountered applications that involve large and sparse matrices. It also enables larger problems to be solved than could previously be handled within a given amount of available memory. Our evaluation method has been embedded in a tabu search algorithm in a sequel to this paper, yielding a method that efficiently matches or improves currently best-known results for instances from widely used benchmark sets having up to 7,000 variables.

Keywords: 0–1 optimisation; unconstrained quadratic programming; metaheuristics; computational efficiency; tabu search.

Reference to this paper should be made as follows: Glover, F. and Hao, J-K. (2010) 'Efficient evaluations for solving large 0–1 unconstrained quadratic optimisation problems', *Int. J. Metaheuristics*, Vol. 1, No. 1, pp.3–10.

Biographical notes: Fred Glover holds the title of Distinguished Professor at the University of Colorado and is the Chief Technology Officer for OptTek Systems, Inc. He has authored or co-authored more than 370 published articles and eight books in the fields of mathematical optimisation, computer science and artificial intelligence. He is the recipient of the distinguished von Neumann Theory Prize, an elected member of the National Academy of Engineering, and has received honorary awards and fellowships from the American Association for the Advancement of Science (AAAS), the NATO Division of Scientific Affairs, the Energy Research Institute (ERI) and numerous other organisations.

Jin-Kao Hao holds a Full Professor position in the Computer Science Department of the University of Angers (France) and is currently the Director of the LERIA Laboratory. His research lies in the design of effective heuristic and metaheuristic algorithms for solving large-scale combinatorial search problems. He is interested in various application areas including bioinformatics, telecommunication networks and transportation. He has co-authored more than 100 peer-reviewed publications in international journals, book chapters and conference proceedings.

1 Introduction

The binary unconstrained quadratic programming problem may be written as:

$$\text{UQP: Minimise } x_o = xQx \quad x \text{ binary} \tag{1}$$

where Q is an n by n matrix of constants and x is an n-vector of binary (0–1) variables. The UQP formulation is notable for its ability to represent a wide range of important problems, including those from social psychology (Harary, 1953), financial analysis (Laughunn, 1970; McBride and Yormark, 1980), computer aided design (Krarup and Pruzan, 1978), traffic management (Gallo et al., 1980; Witsgall, 1975), machine scheduling (Alidaee et al., 1994), cellular radio channel allocation (Chardaire and Sutter, 1994) and molecular conformation (Phillips and Rosen, 1994). Moreover, many combinatorial optimisation problems pertaining to graphs such as determining maximum cliques, maximum cuts, maximum vertex packing, minimum coverings, maximum independent sets and maximum independent weighted sets are known to be capable of being formulated by the UQP problem as documented in papers of Pardalos and Rodgers (1990) and Pardalos and Xue (1994). A review of additional applications and formulations can be found in Kochenberger et al. (2004).

Our purpose in this note is to provide a more effective method for updating evaluations in search processes that operate by complementing (flipping) values of the 0–1 variables, which are used by the current state-of-the-art methods for solving the UQP. We propose a design that improves on the customary mechanism (see, e.g., Glover et al., 1998a; Merz and Freisleben, 2002) by introducing rules for taking advantage of a lower triangular matrix representation and sparse matrix structures. The resulting procedure offers appreciably greater efficiency for the case where the Q matrix contains numerous 0 entries, as typically occurs in large problems. In such applications, our approach requires fewer operations than the method customarily used in previous applications and also allows problems of greater size to be handled within a given computer memory limit.

2 Notation and conventions

Let N = {1, …, n} denote the index set for components of the x vector and the rows and columns of Q. We assume the Q matrix is preprocessed to put it in lower triangular form by redefining (if necessary):

$$Q_{ij} := Q_{ij} + Q_{ji} \text{ followed by } Q_{ji} := 0 \text{ for all i, } j \in N \text{ such that } j > i \tag{2}$$

Thus, for example, a UQP problem for n = 5 with a lower triangular Q matrix has the structure:

	x_1	x_2	x_3	x_4	x_5
x_1	Q_{11}	0	0	0	0
x_2	Q_{21}	Q_{22}	0	0	0
x_3	Q_{31}	Q_{32}	Q_{33}	0	0
x_4	Q_{41}	Q_{42}	Q_{43}	Q_{44}	0
x_5	Q_{51}	Q_{52}	Q_{53}	Q_{54}	Q_{55}

The variables x_1 to x_5 associated with the rows and columns give a convenient way to represent the objective function xQx, which arises by multiplying each Q_{ij} entry shown by the associated row and column variables x_i and x_j and then summing. (Hence, in the present case, the objective function is given by $x_1Q_{11}x_1 + (x_2Q_{21}x_1 + x_2Q_{22}x_2) + (x_3Q_{31}x_1 + x_3Q_{32}x_2 + x_3Q_{33}x_3) + \ldots$.) The indicated preprocessing step to create the lower triangular structure does not change the problem, since $x_ix_j = x_jx_i$.

For our subsequent purposes, we find it useful to factor out the row variables x_i and to write $x_o = xQx$ in the form:

$$x_o = \sum \left(x_i \left(\left(\sum Q_{ij}x_j : j \in N,\ j < i \right) + Q_{ii} \right) : i \in N \right) \tag{3}$$

The diagonal coefficient Q_{ii} can be separated from the off-diagonal coefficients Q_{ij} for $j < i$ in the preceding expression since the fact that x_i is a 0–1 variable which implies $x_iQ_{ii}x_i = x_iQ_{ii}$.

3 Key result

Let x' and x'' represent two binary solutions where x'' is obtained from x' by flipping the value of a single variable x_k from 0 to 1 or from 1 to 0 (according to whether x_k' is 0 or 1). Define $x_o' = x'Qx'$ and $x_o'' = x''Qx''$. Then, the value $\Delta x_o(k) = x_o'' - x_o'$, which depends on the choice of the variable x_k, discloses whether the move that replaces x' by x'' will cause x_o to improve or deteriorate (respectively, decrease or increase) relative to the minimisation objective. The ability to make such an evaluation rapidly affects the efficiency of search methods that takes account of changes in x_o when selecting a variable x_k for the purpose of changing its value.

We first specify a rule for quickly identifying $\Delta x_o(k)$ for each $k \in N$ without undertaking to exploit the sparseness of Q. Then, we describe the implementation of this rule that takes advantage of sparse matrices.

Define RowValue(i) and ColValue(j) as a function of a current solution x' by:

$$\text{RowValue(i)} = \left(\sum Q_{ij}x_j' : j \in N,\ j < i \right) \tag{4}$$

and

$$\text{ColValue}(j) = \left(\sum Q_{ij}x'_i : i \in N,\ i < j\right) \tag{5}$$

By convention, RowValue(i) = 0 when i = 1 and ColValue(j) = 0 when j = n.

During the examination of different alternatives for the choice of x_k, the quantities RowValue(i) and ColValue(j) remain constant for all i, j ∈ N, so that they may be accessed by a simple look up operation without requiring any computation to obtain them. Subsequently, once a particular variable x_k is selected and a move is executed that replaces x_k' by x_k'', the quantities RowValue(i) and ColValue(j) will be updated by an operation that involves a single pass of the indexes from 1 to n and performing a simple addition (or subtraction) for each of them. The rule that achieves this is as follows.

3.1 Evaluation and updating rule

Let δ = 1 if $x_k' = 0$ and let δ = –1 if $x_k' = 1$. Then:

$$\Delta x_o(k) = \delta(\text{RowValue}(k) + \text{ColValue}(k) + Q_{kk}). \tag{6}$$

Upon executing the move that replaces x_k' by x_k'' to yield the solution x″, the new quantities RowValue(i) and ColValue(j), for the specific indexes over which these quantities change, are given by:

$$\text{RowValue}(i) := \text{RowValue}(i) + \delta Q_{ik} \text{ for } i \in N,\ i > k \tag{7}$$

$$\text{ColValue}(j) := \text{ColValue}(j) + \delta Q_{kj} \text{ for } j \in N,\ j < k \tag{8}$$

The use of δ = 1 or –1 is not to literally perform a multiplication, but simply to identify a sign to be attached to a specified quantity.

3.2 Justification of the rule

We rewrite the expression (3) for x_o by decomposing it in the following manner:

$$\sum\left(x_i\left(\left(\sum Q_{ij}x_j : j \in N,\ j < i,\ j \neq k\right) + Q_{ii}\right) : i \in N,\ i \neq k\right) \tag{9.1}$$

$$+\ x_k\left(\sum Q_{ij}x_k : j \in N,\ j < k\right) + Q_{kk}\Big) \tag{9.2}$$

$$+\ \sum\left(x_i\left(Q_{ik}x_k : k < i\right) + Q_{ii}\right) : i \in N\Big) \tag{9.3}$$

Note that (9.3) can also be written as:

$$+\ x_k\left(\sum\left(Q_{ik}x_i : i \in N,\ i > k\right)\right) + \sum\left(Q_{ii}x_i : i \in N,\ i > k\right) \tag{9.4}$$

Given that $x_i'' = x_i'$ for all $i \neq k$, it follows that $\Delta x_o(k)$ can be reduced to:

$$\left(x_i'' - x_i'\right)\left(\left(\sum Q_{ij}x_k : j \in N,\ j < k\right) + Q_{kk}\right) + \sum((Q_{ik}x_i : i \in N,\ i > k))) \tag{10}$$

From the definitions, we see that $\delta = x_i'' - x_i'$ and the summation terms within (10) (which exclude Q_{kk}) equal RowValue(k) and ColValue(k). This establishes the validity of the expression (6) for $\Delta x_o(k)$. The updated forms of RowValue(i) and ColValue(j) given in (7) and (8) follow by a corresponding analysis.

4 Exploiting sparseness

To exploit sparseness using the expressions (7) and (8) for updating RowValue(i) and ColValue(j), we introduce the following data structures: 'successive indexes h = 1, 2, …, hLast are assigned to the non-zero off-diagonal entries of Q, where each such non-zero Q_{ij} for i > j is recorded by setting Q(h) = Q_{ij} and simultaneously recording Row(h) = i and Col(h) = j. The diagonal Q_{ii} values for i ∈ N are recorded separately by setting $Q_o(i) = Q_{ii}$.

This data structure is additionally augmented to include linked lists that make it possible to access the entries Q(h) either of two ways: by row or by column. For this, we introduce two arrays RowFirst(i) and ColFirst(j) for i, j ∈ N, which are initialised by setting all their entries to 0. Then, each time a new entry Q(h) is entered via the input data, we refer to the associated values i = Row(h) and j = Col(h) and set:

RowNext(h) = RowFirst(i)
RowFirst(i) = h
ColNext(h) = ColFirst(j)
ColFirst(j) = h.

By making use of these records, upon the conclusion of the input process, all non-zero entries of row i can be traced by the sequence:

h = RowFirst(i)
While h ≠ 0:
    Q(h) is a non-zero entry Q_{ij} of Q, where i = Row(h) and j = Col(h)
    h := RowNext(h)
EndWhile

All non-zero entries of column j can be traced in a similar manner by replacing RowFirst(i) with ColFirst(j) and replacing RowNext(h) with ColNext(h).

The entries of a given row or column accessed in this fashion will be encountered in the reverse order from the sequence of the input data. We note that this order is irrelevant from the standpoint of applying the evaluation and update rule embodied in the expressions (6), (7) and (8).

If the input data do not initially provide the entries of Q in lower triangular form, then an associated preprocessing operation is required as previously noted. The extra memory for such a step depends on the protocol used for entering the data (e.g., whether all entries of each row or of each column appear in a single block).

The use of additional memory can be avoided by performing a further step for each non-zero entry Q_{ij} encountered, as follows.

a If $i > j$, the linked sequence of h values starting with h = RowFirst(j) (and proceeding with h := RowNext(h)) is scanned to see if i = Col(h) is encountered as a column entry.
   1 if i = Col(h) is found in this fashion, augment Q(h) by setting $Q(h) := Q(h) + Q_{ij}$
   2 otherwise, if i is not found in the array Col(h) for the values of h traced, then record Q_{ij} as a new Q(h) value by setting:

$$hLast := hLast + 1$$
$$Row(hLast) = j \text{ and } Col(hLast) = i.$$

b If $i < j$, likewise check whether a value for Q_{ij} was previously recorded (since it may have been produced by the process just described). In this case, the trace is initiated starting with h = RowFirst(i) (and proceeding with h := RowNext(h)) to see if j = Col(h) is encountered.
   1 if j = Col(h) is found in this fashion, augment Q(h) by setting $Q(h) := Q(h) + Q_{ij}$
   2 otherwise, if j is not found in the array Col(h) for the values of h traced, then record Q_{ij} as a new Q(h) value by setting:

$$hLast := hLast + 1$$
$$Row(hLast) = i \text{ and } Col(hLast) = j.$$

By means of this data structure, the updating specified in (7) to update RowValue(i) for $i > k$ proceeds as follows:

```
h = ColFirst(k)
While h ≠ 0:
        RowValue(i) := RowValue(i) + δQ_ik
        h := ColNext(h)
EndWhile
```

An analogous rule updates ColValue(j) for $j < k$ as specified in (8), interchanging 'Row' and 'Col' and replacing (i) with (j) and Q_{ik} with Q_{kj}.

Although these processes are somewhat more complex than those customarily used to perform updating operations in solving 0–1 UQP problems, the potential for solving large and sparse problems with significantly greater efficiency makes them attractive for inclusion within future search methods for UQP.

5 Concluding remarks

The fast evaluation strategy introduced here has been integrated in a recent tabu search algorithm for the binary unconstrained quadratic programming problem (Glover et al., 2009). Thanks to this strategy, the algorithm is able to efficiently examine, at each iteration, a large number of neighbouring solutions defined by the one-flip move. Combined with an extended memory-based diversification strategy, the proposed algorithm proves to be highly effective in solving a range of benchmark instances from the literature. For example, for the well-known UQP instances (up to 2,500 variables) introduced in Glover et al. (1998b) and Beasley (1998) and used by many published

papers, this algorithm reaches the best-known objective values on average in less than one minute on a PC with Pentium 2.66 GHz CPU and 512 M RAM, representing a decrease of at least 40% compared with previous implementations. Moreover, tested on the set of 21 large instances with 3,000 to 7,000 variables introduced in Palubeckis (2004, 2006), the algorithm is able to equal or even improve the previously best results.

Furthermore, we note that the one-flip move evaluation introduced in this note can also contribute to the specification of a fast evaluation strategy for more sophisticated moves such as two-flip moves. Such a perspective is particularly useful for establishing complementary and combined neighbourhood relations that are indispensable for solving larger and more diverse classes of UQP instances.

Finally, a more explicit analysis of improvements in computation time and memory, as described in the Glover et al.'s (2009) paper, constitutes an interesting direction for future work.

Acknowledgements

The work is partially supported by a 'Chaire d'excellence' from 'Pays de la Loire' Region (France)' and regional MILES (2007–2009) and RaDaPop projects (2008–2011). We are grateful for the comments by a referee that have improved the exposition of this note.

References

Alidaee, B., Kochenberger, G. and Ahmadian, A. (1994) '0–1 quadratic programming approach for the optimal solution of two scheduling problems', *International Journal of Systems Science*, Vol. 25, pp.401–408.

Beasley, J.E. (1998) *Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem*, Technical report, Management School, Imperial College, UK.

Chardaire, P. and Sutter, A. (1994) 'A decomposition method for quadratic zero–one programming', *Management Science*, Vol. 41, No. 4, pp.704–712.

Gallo, G., Hammer, P. and Simeone, B. (1980) 'Quadratic knapsack problems', *Mathematical Programming*, Vol. 12, pp.132–149.

Glover, F., Kochenberger, G., Alidaee, B. and Amini, M. (1998a) 'Tabu search with critical event memory: an enhanced application for binary quadratic programs', in S. Voss, S. Martello, I.H. Osman and C. Roucairol (Eds.): *Meta-Heuristics – Advances and Trends in Local Search Paradigms for Optimization*, pp.83–109, Kluwer Academic Publishers.

Glover, F., Kochenberger, G.A. and Alidaee, B. (1998b) 'Adaptive memory tabu search for binary quadratic programs', *Management Science*, Vol. 44, No. 3, pp.336–345.

Glover, F., Lü, Z. and Hao, J.K. (2009) *Diversification-Driven Tabu Search for Unconstrained Binary Quadratic Problems*, Research report, LERIA, Université d'Angers.

Harary, F. (1953) 'On the notion of balanced of a signed graph', *Michigan Mathematical Journal*, Vol. 2, pp.143–146.

Kochenberger, G., Glover, F., Alidaee, B. and Rego, C. (2004) 'A unified modeling and solution framework for combinatorial optimization problems', *OR Spectrum*, Vol. 26, pp.237–250.

Krarup, J. and Pruzan, A. (1978) 'Computer aided layout design', *Mathematical Programming Study*, Vol. 9, pp.75–94.

Laughunn, D.J. (1970) 'Quadratic binary programming', *Operations Research*, Vol. 14, pp.454–461.

McBride, R.D. and Yormark, J.S. (1980) 'An implicit enumeration algorithm for quadratic integer programming', *Management Science*, Vol. 26, pp.282–296.

Merz, P. and Freisleben, B. (2002) 'Greedy and local search heuristics for unconstrained binary quadratic programming', *Journal of Heuristics*, Vol. 8, No. 2, pp.197–213.

Palubeckis, G. (2004) 'Multistart tabu search strategies for the unconstrained binary quadratic optimisation problem', *Annals of Operations Research*, Vol. 131, pp.259–282.

Palubeckis, G. (2006) 'Iterated tabu search for the unconstrained binary quadratic optimization problem', *Informatica*, Vol. 17, No. 2, pp.279–296.

Pardalos, F. and Xue, J. (1994) 'The maximum clique problem', *The Journal of Global Optimization*, Vol. 4, pp.301–328.

Pardalos, P. and Rodgers, G.P. (1990) 'Computational aspects of a branch and bound algorithm for quadratic zero–one programming', *Computing*, Vol. 45, pp.131–144.

Phillips, A.T. and Rosen, J.B. (1994) 'A quadratic assignment formulation of the molecular conformation problem', *Journal of Global Optimization*, Vol. 4, pp.229–241.

Witsgall, C. (1975) *Mathematical Methods of Site Selection for Electronic System (EMS)*, NBS Internal report.