Polynomial unconstrained binary optimisation – Part 1

Fred Glover*

OptTek Systems, Inc.,
2241 17th St., Boulder, CO 80302, USA
E-mail: glover@opttek.com

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

Gary Kochenberger

School of Business Administration,
University of Colorado at Denver,
Denver, CO 80217, USA
E-mail: gary.kochenberger@cudenver.edu

Abstract: The class of problems known as quadratic zero-one (binary) unconstrained optimisation has provided access to a vast array of combinatorial optimisation problems, allowing them to be expressed within the setting of a single unifying model. A gap exists, however, in addressing polynomial problems of degree greater than 2. To bridge this gap, we provide methods for efficiently executing core search processes for optimisation problems in the general polynomial unconstrained binary (PUB) domain. A variety of search algorithms for quadratic optimisation can take advantage of our methods to be transformed directly into algorithms for problems where the objective functions involve arbitrary polynomials.

In this Part 1 paper, we give fundamental results for carrying out the transformations. We also describe coding and decoding procedures that are relevant for efficiently handling sparse problems, where many coefficients are 0, as typically arise in practical applications. In a sequel to this paper, Part 2, we provide special algorithms and data structures for taking advantage of the basic results of Part 1. We also disclose how our designs can be used to enhance existing quadratic optimisation algorithms.

Keywords: zero-one optimisation; unconstrained polynomial optimisation; metaheuristics; computational efficiency.

Reference to this paper should be made as follows: Glover, F., Hao, J-K. and Kochenberger, G. (2011) 'Polynomial unconstrained binary optimisation – Part 1', *Int. J. Metaheuristics*, Vol. 1, No. 3, pp.232–256.

Copyright © 2011 Inderscience Enterprises Ltd.

Biographical notes: Fred Glover is a 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 400 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. He is 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 Miller Institute of Basic Research in Science and numerous other organisations.

Jin-Kao Hao is a Full Professor 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.

Gary Kochenberger is a Full Professor of Decision Science at the University of Colorado at Denver and is a Co-director of the Decision Science programme. His research focuses on designing and testing metaheuristics methods for large scale optimisation problems. He has co-authored more than 70 refereed papers and three books.

1 Introduction

1.1 Problem representation

We formulate the polynomial unconstrained binary (PUB) optimisation problem as

$$PUB: \underset{x \text{ binary}}{Minimise}\ x_o = c_o + \sum \left(c_p F_p : p \in P \right)$$

The vector $x = (x_1,x_2,\ldots,x_n)$ consists of binary variables, $x_i \in \{0,1\}$ for $i \in N = \{1,\ldots,n\}$, and the coefficients c_p for $p \in P = \{1,\ldots,p_o\}$ are non-zero scalars. F_p is a product of components of the x vector given by

$$F_p = \Pi \left(x_i : i \in N_p \right) \text{ where } N_p \subset N.$$

Each variable x_i in the product defining F_p appears only once, noting that $x_i^h = x_i$ for x_i binary, which renders powers h of x_i other than $h = 1$ irrelevant.

A seemingly more general formulation arises by replacing the sets N_p, $p \in P$ by vectors N_p^o, $p \in P^o$ and writing $F_p = x_{i1}x_{i2}\ldots x_{ih}$ for $\left(i_1, i_2, \ldots, i_h\right) = N_p^o$, which allows different coefficients c_p for different permutations of the same set of indexes. However, as noted in the following observation, such a formulation in terms of permutations can be readily converted into a formulation in terms of sets, with the advantage of reducing the number of non-zero coefficients (and thereby increasing the sparsity of the problem representation).

Remark 1: In a polynomial representation based on permutations, where two permutations $N_p^o = (i_1, i_2, \ldots, i_h)$ and $N_q^o = (j_1, j_2, \ldots, j_h)$, are over the same set of indexes, and the associated costs c_p^o and c_q^o are both non-zero, an equivalent problem results by redefining $c_p^o := c_p^o + c_q^o$ and $c_q^o = 0$, thus eliminating the term for the vector N_q^o.

The validity of the remark follows simply from the fact that the product $x_{i1}x_{i2}\ldots x_{ih}$ has the same value as the product $x_{j1}x_{j2}\ldots x_{jh}$. By summing the costs for different permutations as indicated, the remark gives a basis for a preprocessing step enabling any permutation-based formulation of PUB to be converted into the form shown. Such a preprocessing step may also be viewed as equivalent to producing a restricted permutation formulation where each permutation $N_p^o = (i_1, i_2, \ldots, i_h)$ has the *ascending index property* $i_1 < \ldots < i_h$.

To see how the PUB formulation relates to more classical formulations that embody the ascending index property, consider the cubic polynomial given by $\sum(q_i x_i : i \in N) + \sum\sum(q_{ij}x_i x_j : i < j \in N) + \sum\sum\sum(q_{ijk}x_i x_j x_k : i < j < k \in N)$. Let $N(1) = \{i \in N: q_i \neq 0\}$, $N(2) = \{(i,j): i < j \in N: q_{ij} \neq 0\}$, $N(3) = \{(i,j,k): i < j < k \in N: q_{ijk} \neq 0\}$. Then the representation is completed by assigning the index p the consecutive integer values $p = 1$ to $p_o = |N(1)| + |N(2)| + |N(3)|$, and letting each consecutive p take the associated value q_i or q_{ij} or q_{ijk}, together with $N_p = \{i\}$ or $\{i,j\}$ or $\{i,j,k\}$, for $i \in N(1)$, $(i,j) \in N(2)$, $(i,j,k) \in N(3)$.

The next remark is useful to facilitate certain operations of our method.

Remark 2: We assume $P_j = \{j\}$ for $j \in N$ without regard to the value of c_j, thus providing an exception to the rule of only including terms with non-zero coefficients in the PUB representation.

1.1.1 Illustration of the PUB representation

Consider the PUB problem whose objective function is given by

$$x_o = 7 + 5x_1 + 2x_2 - 3x_2^2 - 4x_3x_1 + 5x_1x_2 - 2x_2x_1 + 3x_1^3x_2x_3.$$

First, since $x_2^2 = x_2$, $x_1^3 = x_1$ *and* $x_1x_2 = x_2x_1$, we can re-write x_o in ascending index notation, including only the non-zero coefficients, as

$$x_o = 7 + 5x_1 - x_2 - 4x_1x_3 + 3x_1x_2 + 3x_1x_2x_3.$$

By Remark 2, we include the x_3 term in the representation, even though it has a 0 coefficient, to give

$$x_o = 7 + 5x_1 - x_2 + 0x_3 - 4x_1x_3 + 3x_1x_2 + 3x_1x_2x_3.$$

Assigning indexes $p = 1$ to 6 to the terms in sequence, we identify the indexes of the variables in these terms by

$$N_1 = \{1\},\ N_2 = \{2\},\ N_3 = \{3\},\ N_4 = \{1,3\},\ N_5 = \{1,2\},\ N_6 = \{1,2,3\}$$

The cost coefficients associated with these terms, including the constant term c_o, are given by

$$c_o = 7,\ c_1 = 5,\ c_2 = -1,\ c_3 = 0,\ c_4 = -4,\ c_5 = 3,\ c_6 = 3.$$

(Note that while we assume the indexes of any set N_p are in ascending order, we do not assume the sets themselves are organised in a lexicographic ascending order. Hence, in the example it is acceptable to give a smaller p index to the set {1,3} than to the set {1,2}.)

1.2 Applications and motivation

The special case where the polynomial objective of PUB is a quadratic function (hence the polynomial has a degree of 2) has been widely studied. In this case, PUB already represents a broad range of important problems, including those from social psychology (Harary, 1953), financial analysis (Laughunn, 1970; McBride and Yormack, 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 PUB in the quadratic case 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).

The more general PUB formulation given here is of interest for its ability to encompass a significantly expanded range of problems. The cubic case, for example, permits PUB to represent the important class of satisfiability problems known as 3-SAT, and offers the advantage of providing a representation whose size does not depend on the number of logical clauses, which stands in contrast to the case for customary binary integer programming formulations of 3-SAT (see Kochenberger, 2010). The availability of procedures for efficiently handling and updating move evaluations for instances of PUB involving polynomials of degree greater than 2, as identified in the following sections, provides an impetus to uncover additional problems that can be usefully given binary polynomial formulations.

Our procedures, which apply to moves that flip (complement) the values of one or more variables x_j in progressing from one solution to another, constitute a generalisation of procedures for generating 1-flip moves described in Glover et al. (1998) and extended to 2-flip and multi-flip moves Glover and Hao (2010a, 2010b). Important recent contributions of a similar nature for the quadratic problem are provided in Hanafi et al. (2010). A principle outcome of our development is that a variety of search methods which currently incorporate procedures to evaluate flip moves for the quadratic problem can replace these procedures by the methods described here, thereby producing methods capable of solving general PUB problems without any other changes in their structure.

2 Preliminary relationships

We start by making some basic observations that directly generalise observations given in Glover and Hao (2010a, 2010b). These observations provide a basis for a more encompassing framework given in the next section which is particularly useful for exploiting sparsity.

Let x' and x'' represent two binary solutions and define

$$x'_o = \sum\left(c_p F'_p : p \in P\right), \text{ where } F'_p = \Pi\left(x'_k : k \in N_p\right)$$

$$x''_o = \sum\left(c_p F''_p : p \in P\right), \text{ where } F''_p = \Pi\left(x''_k : k \in N_p\right).$$

$$\Delta x_o = x''_o - x'_o$$

The objective function change Δx_o thus discloses whether the transition (move) from x' to x'' will cause x_o to improve or deteriorate (respectively, decrease or increase) relative to the minimisation objective.

We identify the variables x_i that are complemented in going from the solution x' to the solution x'', and the subsets for which $x''_i = 1$ and 0 by defining

$$N^c = \left\{i \in N : x''_i = 1 - x'_i\right\}$$

$$N^c(1) = \left\{i \in N^c : x''_i = 1\right\}$$

$$N^c(0) = \left\{i \in N^c : x''_i = 0\right\}$$

$$P(M) = \left\{p \in P : N_p \subset M\right\}, \text{ where } M \text{ is an arbitrary subset of } N.$$

Observation 1: If $x'_i x''_i = 0\left(x'_i = 0 \text{ or } x''_i = 0\right)$ for each $i \in N$, then

$$\Delta x_o = \sum\left(c_p : p \in P\left(N^c(1)\right)\right) - \sum\left(c_p : p \in P\left(N^c(0)\right)\right)$$

Proof: The identity $\Delta x_o = \sum\left(cpF''_p : p \in P\right) - \sum\left(c_p F'_p : p \in P\right)$ may be rewritten as $\sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c\right)\right) + \sum\left(c_p\left(F''_p - F'_p\right) : p \in P - P\left(N^c\right)\right)$. If $p \in P - P(N^c)$, then N_p must contain at least one k such that $x'_k = x''_k = 0$, and hence $F''_p = F'_p = 0$. Consequently, $\Delta x_o = \sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c\right)\right)$.

We rewrite the latter as

$$\sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c(1)\right)\right) + \sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c(0)\right)\right) +$$
$$\sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c\right) - P\left(N^c(1)\right) - P\left(N^c(0)\right)\right).$$

For each p in the range of this latter summation, N_p is not a subset of $N^c(1)$ and also not a subset of $N^c(0)$, which implies N_p must contain a pair $\{j,k\}$ such that $x''_j = 1,\ x'_j = 0$ and $x''_k = 0,\ x'_k = 1.$

Thus, $F''_p = F'_p = 0$ for each of the terms of this latter summation, and we are left with $\Delta x_o = \sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c(1)\right)\right) + \sum\left(c_p\left(F''_p - F'_p\right) : p \in P\left(N^c(0)\right)\right)$. But $F''_p = 1$ and $F'_p = 0$ for $p \in P(N^c(1))$ while $F''_p = 0$ and $F'_p = 1$ for $p \in P(N^c(0))$. This completes the proof.

In each of the following two corollaries, we make the more restrictive assumption that $x' = 0.$ Let $N'' = \left\{i \in N : x''_i = 1\right\}\left(\text{hence } P\left(N''\right) = \left\{p \in P : N_p \subset N''\right\}\right)$ to identify those sets N_p such that $x''_i = 1$ for all $i \in N_p$. Thus, if $N'' = \left\{i_1,\ldots,i_h\right\},$ then $P(N'')$ is the index set for all those sets N_p composed of one or more of the elements $i_1,\ldots,i_h$.

Corollary 1.1: If $x' = 0,$ then $\Delta x_o = \sum\left(c_p : p \in P\left(N''\right)\right)$

Proof: The result follows directly from Observation 1, by noting that for $x' = 0$ we have $N'' = N^c(1)$ and $N^c(1) = \left\{i \in N^c : x''_i = 1\right\}$. At the same time, the definition $N^c(0) = \left\{i \in N^c : x''_i = 0\right\},$ which implies $N^c(0) = \left\{i \in N^c : x'_i = 1\right\},$ showing that $N^c(0)$ is empty.

To simplify the statement of the next result, it is useful to isolate the coefficients c_p of the product terms F_p that refer only to a single variable x_j. As previously noted, we suppose $N_p = \{p\}$ for $p \in N$, although $c_p = 0$ is possible for some of these indexes. Hence, the product term F_p for $p \in N$ is the single variable term $F_p = x_p$. If we denote indexes $p \in N$ instead by $j \in N$ to conform to the practice of referring to variables x_j for $j \in N$, our indexing convention identifies the 'x_j component' of the objective function x_o to be just $c_j x_j$.

Corollary 1.2: If $x' = 0,$ and x'' results from x' by flipping the single variable x_j, then $\Delta x_o = c_j$.

Proof: This result is an immediate consequence of Corollary 1.1 where $N'' = \{j\}$. Corollary 2 can also be established directly by noting that the solution $x'' = x' + x''_j e_j$ yields $F''_p = 0$ for all $p \in P$ except for $p = j$, since each term F''_p other than for $p = j$ contains a variable x_k such that $x''_k = x'_k = 0.$ Hence, Δx_o reduces to $c_j x''_j - c_j x'_j,$ which is just c_j, as stipulated.

In the context of Corollaries 1.1 and 1.2, c_j gives the change in x_o that results from flipping a single variable x_j, while $\sum\left(c_p : p \in P(N'')\right)$ is the change that results from flipping all variables x_i for $i \in N''$. Hence, Corollary 1.2 gives an evaluation of a 1-flip move and Corollary 1.1 gives an evaluation of a q-flip move by letting the set N'' represent the indexes for a selected set of q variables that are flipped from 0 to 1.

3 Exploiting and updating problem transformations

The preceding corollaries have an important implication: provided we start from a solution $x' = 0$, the amount of effort to evaluate a move involving any number of flips, from 1 to q, is the same for any polynomial of degree d for $d \geq q$ (the degree d may be defined by $d = \max(|N_p|: p \in P)$. Thus, by this stipulation, the work to evaluate a 1-flip is the same for all polynomials, the work to evaluate a 2-flip is the same for all polynomials of degree 2 or greater, and so on. It also follows that the work to evaluate a q-flip for a polynomial of degree $d \geq q$ only requires a single addition operation beyond the work to evaluate a q-flip for a polynomial of degree $d = q - 1$. Consequently, a 3-flip in a polynomial of degree 3 or larger can be evaluated by using only one addition operation beyond that required to evaluate a 3-flip in a polynomial of degree 2.

Corollaries 1.1 and 1.2 therefore provide a natural motivation to manipulate the formulation of PUB so that a current solution x' may always be treated as if it were the 0 solution. To do this we use the common device of transforming the x vector into another binary vector y by complementing selected components of x; in this case, specifically complementing those components of x such that $x'_j = 1$, thus causing the assignment $x = x'$ to yield a corresponding assignment $y = y'$ for which $y' = 0$.

For greater precision, we refer to the formulation PUB as PUB(x), and consider the alternative equivalent formulation PUB(y) based on the transformed vector y. To express PUB(y), let

$$N'(1) = \left\{ i \in N : x'_i = 1 \right\} \text{ and } N'(0) = \left\{ i \in N : x'_i = 0 \right\}$$

and define the relationship between y and x by

$$y_i = 1 - x_i,\ i \in N'(1)$$

$$y_j = x_j,\ i \in N'(0)$$

This transformation of variables causes the objective for PUB(y) to include different terms than the objective of PUB(x). We demonstrate how this occurs by decomposing the transition from the formulation PUB(x) to the formulation PUB(y) into a series of 1-flip steps, each consisting of implementing the complementation operation for a single variable, i.e., for a single index $i \in N'(1)$, and then repeating until the operation has been completed for all $i \in N'(1)$.

The amended terms c_pF_p for PUB(y) require altering the identity of the set P, in order that it may continue to refer strictly to sets N_p such that $c_p \neq 0$ with the exception that we include the sets $N_j = \{j\}$ for $j \in N$ even if $c_j = 0$ as noted earlier. In other words, the methods we describe will result in changing P by identifying certain terms c_pF_p for which the coefficient c_p for $p > n$ will change from $c_p = 0$ to $c_p \neq 0$, and certain other terms for which the coefficient c_p for $p > n$ will change from $c_p \neq 0$ to $c_p = 0$. Thus, in the former case p will not belong to P, and will be added to P (by setting $P := P \cup \{p\}$) while in the latter case p will be dropped from P (by setting $P := P - \{p\}$).

The maintenance of P in this manner, so that it always refers to terms having non-zero coefficients for $p > n$, is important for exploiting sparsity, which is a

characteristic feature of many PUB problems, particularly those of moderate to large size. To treat sparsity effectively we implicitly refer to an additional set of indexes, denoted by P^α, which includes relevant indexes p for which $c_p = 0$, i.e., specifying that the collection $\{N_p: p \in P^\alpha\}$ consists of all subsets N_p of N containing from 1 to d indexes of N. Later, we provide a compact coding mechanism that makes it possible to identify elements of P^α that are relevant for algorithmic updates without needing to rely on searches to carry out this identification.

To broaden the generality of our results, we introduce a special set N_o and a corresponding 'product term' F_o associated with the objective function variable x_o, where we stipulate that $N_o = \varnothing$. By the standard convention that the product of variables over the empty set equals 1, we have $F_o = 1$ (applying the definition $F_p = \Pi(x_k: k \in N_p)$ to the case where $N_p = N_o$). This yields $c_oF_o = c_o$, and hence c_oF_o is just the constant term associated with the objective function x_o. These conventions allow us to express changes in x_o using the same notation employed to express changes in general terms of the form c_pF_p. Consequently, we understand that P^α includes the index $p = 0$, in order to include reference to x_o and c_o. The relevance of these stipulations will become clear in an illustration subsequently provided.

To address the 1-flip case we denote the variable that is flipped by x_j, hence yielding $y_j = 1 - x_j$. Then we define the following for each $j \in N$:

$$P(j) = \left\{ p \in P : j \in N_p \right\}$$

$$p[j] = \text{ the unique index in } P^\alpha \text{ such that } N_{p[j]} = N_p - \{j\}$$

$$\left(\textit{hence, } F_{p[j]} = \Pi\left(x_k : k \in N_p - \{j\} \right), \text{ and } c_{p[j]} = 0 \text{ if } p[j] \notin P \right).$$

$$F_p[j] = y_j F_{p[j]} \ \left(\text{hence, } F_p[j] \text{ is the same as } F_p \text{ except that } y_j \text{ replaces } x_j \right).$$

We observe that $P(j)$, which identifies the index set for all sets N_p that contain j, is effectively a special case of $P(M) = \{p \in P: N_p \subset M\}$, by taking $M = \{j\}$.

3.1 Illustration of the set P(j)

The set $P(j)$ plays a pivotal role in several parts of our development, and hence we illustrate its composition by reference to the example polynomial used earlier to illustrate the notational conventions underlying the PUB representation. After applying these conventions the polynomial took the form

$$x_o = 7 + 5x_1 - x_2 + 0x_3 - 4x_1x_3 + 3x_1x_2 + 3x_1x_2x_3.$$

which gave rise to the sets

$$N_1 = \{1\},\ N_2 = \{2\},\ N_3 = \{3\},\ N_4 = \{1,3\},\ N_5 = \{1,2\},\ N_6 = \{1,2,3\}.$$

Then the sets $P(j)$ for $j \in N = \{1,2,3\}$ are given by

$$P(1) = \{1,4,5,6\},\ P(2) = \{2,5,6\},\ P(3) = \{3,4,6\}.$$

(The terms $p[j]$ and $F_p[j]$ are readily understood by reference to the composition of $P(j)$.)

Observation 2: Flipping x_j to replace x_j by $y_j = 1 - x_j$ produces the following changes for each index $p \in P(j)$:

$$c_{p[j]} := c_{p[j]} + c_p \left(\text{changing } c_{p[j]}F_{p[j]} \text{ to become } \left(c_{p[j]} + c_p\right)F_{p[j]}\right).$$

$$c_p := -c_p$$

$$F_p := F_p[j] \left(\text{changing } c_p F_p \text{ to become } c_p F_p[j] \text{ for the new } c_p \text{ value}\right).$$

Moreover, these changes are independent, so that the change for one index $p \in P(j)$ does not affect the change for another $p \in P(j)$.

Proof: The term c_pF_p for $p \in P(j)$ can be written as

$$c_p F_p = x_j \Pi\left(x_k : k \in N_p - \{j\}\right) = x_j c_p F_{p[j]}.$$

Flipping x_j, which corresponds to substituting $1 - y_j$ for x_j thus transforms c_pF_p into $(1 - y_j)c_pF_{p[j]} = c_pF_{p[j]} - y_jc_pF_{p[j]} = c_pF_{p[j]} - c_pF_p[j]$.

Hence, the term $(c_{p[j]} + c_p)F_{p[j]}$ replaces the term $c_{p[j]}F_{p[j]}$, defined over the index set $N_{p[j]} = N_p - \{j\}$, and after setting $c_p := -c_p$ the term $c_pF_p[j]$ replaces the previous term c_pF_p, defined over the same index set N_p.

The independence of these changes follows from the fact that each set N_p for $p \in P(j)$ is unique and hence each set $N_{p[j]} = N_p - \{j\}$ is also unique.

It is important to observe that some or all of the indexes $p[j]$ may not belong to P (as occurs when $c_{p[j]} = 0$). If $p[j] \notin P$, then except for the special case where N_p contains the index of a single variable, $p \in P$ implies $c_p \neq 0$, and hence the new coefficient $c_{p[j]} := c_{p[j]} + c_p$ must be non-zero. This compels P to be enlarged for the representation PUB(y) by setting $P := P \cup \{p[j]\}$. On the other hand, if $p[j] \in P$, then it is possible that the new value $c_{p[j]} + c_p$ of $c_{p[j]}$ may become 0, and in this case P must be reduced by setting $P := P - \{p[j]\}$. Again, we later give processes for handling such operations efficiently.

The update produced by Observation 2 is completed by redefining x to be y (hence, redefining $x_j := 1 - x_j$) so that we may treat the current solution as $x' = 0$, and apply Observation 2 again to repeat the process. A record of the true solution x is kept throughout these operations, but the algorithm does not have to know this solution, and by means of the currently updated formulation only 'sees' a current collection of terms c_pF_p and their associated sets N_p for $p \in P$ (referring to the current P).

We clarify these comments by means of the following example. We continue to employ the convention $N_p = \{p\}$ for $p \in N$, including the case where p may not belong to P (i.e., when $c_p = 0$).

3.2 Illustration of Observation 2

Let $j = 3$ denote the index of the variable x_j that is flipped, and let

$$P(j) = P(3) = \{3, 30, 40, 50, 60\}$$

where[1]

$$N_3 = \{3\},\ N_{30} = \{3,7\},\ N_{40} = \{1,2,3\},\ N_{50} = \{2,3,5\},\ N_{60} = \{1,2,3,5,7\}.$$

From the definition $N_{p[j]} = N_p - \{j\}$, we have

$$N_{3[3]} = \{\varnothing\},\ N_{30[3]} = \{7\},\ N_{40[3]} = \{1,2\},\ N_{50[3]} = \{2,5\}, N_{60[3]} = \{1,2,5,7\}.$$

In accordance with conventions mentioned earlier, we refer to the special case $N_{p[p]}$, here $N_{3[3]}$, by $N_o = \{\varnothing\}$, taking the index 3[3] to be 0. To assign specific p indexes to the remaining indexes $p[j] = p[3]$, suppose

$$30[3] = 7,\ 40[3] = 15,\ 50[3] = 25 \text{ and } 60[3] = 55.$$

Note that 30[3] = 7 satisfies the convention $N_7 = \{7\}$, i.e., $N_p = \{p\}$ for $p \le n$, given that $N_{30[3]} = \{7\}$. The remaining indexes p = 15, 25 and 55 are chosen simply for purposes of illustration.

Going through the sets N_p and $N_{p[3]}$ in the sequence given by $P(3) = \{3,30,40,50,60\}$ generates the following changes. The symbol '$\to$' is used to denote 'becomes', and the variables are listed in ascending index order. The first case below, for $N_3 = \{3\}$, utilises the convention that $F_o = 1$. The stipulations for this case may be understood more clearly by comparing them to the stipulations for the second case where $N_{30} = \{3,7\}$.

- for $N_3 = \{3\}$ and $N_{3[3]} = N_o = \{\varnothing\}$

$$c_o F_o = c_o \to (c_o + c_3) \to c_o\,(c_o := c_o + c_3)$$

$$c_3 F_3 = c_3 x_3 \to -c_3 y_3 \to c_3 y_3\,(c_3 := -c_3)$$

- for $N_{30} = \{3,7\}$ and $N_{30[3]} = N7 = \{7\}$

$$c_7 F_7 = c_7 x_7 \to (c_7 + c_{30}) x_7 \to c_7 x_7\,(c_7 := c_7 + c_{30})$$

$$c_{30} F_{30} = c_{30} x_3 x_7 \to -c_{30} y_3 x_7 \to c_{30} y_3 x_7\,(c_{30} := -c_{30})$$

- for $N_{40} = \{1,2,3\}$ and $N_{40}[3] = N_{15} = \{1,2\}$

$$c_{15} F_{15} = c_{15} x_1 x_2 \to (c_{15} + c_{40}) x_1 x_2 \to c_{15} x_2 x_5\,(c_{15} := c_{15} + c_{40})$$

$$c_{40} F_{40} = c_{40} x_1 x_2 x_3 \to -c_{50} x_1 x_2 y_3 \to c_{40} x_1 x_2 y_3\,(c_{40} := -c_{40})$$

- for $N_{50} = \{2,3,5\}$ and $N_{50[3]} = N_{25} = \{2,5\}$

$$c_{25} F_{25} = c_{25} x_2 x_5 \to (c_{25} + c_{50}) x_2 x_5 \to c_{25} x_2 x_5\,(c_{25} := c_{25} + c_{50})$$

$$c_{60} F_{60} = c_{50} x_2 x_3 x_5 \to -c_{50} x_2 y_3 x_5 \to c_{50} x_2 y_3 x_5\,(c_{50} := -c_{50})$$

- for $N_{60} = \{1,2,3,5,7\}$ and $N_{60[3]} = N_{55} = \{1,2,5,7\}$

$$c_{55} F_{55} = c_{55} x_1 x_2 x_5 x_7 \to (c_{55} + c_{60}) x_1 x_2 x_5 x_7 \to c_{55} x_1 x_2 x_5 x_7\,(c_{55} := c_{55} + c_{60})$$

$$c_{60} F_{60} = c_{60} x_1 x_2 x_3 x_5 x_7 \to -c_{60} x_1 x_2 y_3 x_5 x_7 \to c_{60} x_1 x_2 y_3 x_5 x_7\,(c_{60} := -c_{60})$$

Once all changes indicated in the preceding illustration are implemented, y_3 is redefined to be x_3, allowing the process to be applied anew to flip additional variables. The illustration demonstrates how the conventions $N_o = \{\varnothing\}$ and $F_o = 1$ permit Observation 2 to identify the update to the objective function constant c_o. By this device, Observation 2 subsumes Corollary 1.2. However, the simpler statement of Corollary 1.2 is convenient for evaluating 1-flips.

3.3 Update to handle multiple flips simultaneously

We now consider the generalisation of Observation 2 that gives the form of the update for performing an arbitrary collection of flips. Let M be the index set for the variables flipped; i.e., $M = \{j \in N: y_j = 1 - x_j\}$. We will generate a collection of terms from M and those sets N_p having a non-empty intersection with M, i.e., for which the set M_p, defined by $M_p = N_p \cap M$, is non-empty. Each term produced from pairing M with a set N_p derives from expanding the current term c_pF_p as a result of complementing the elements of M (which is the reason for stipulating $M_p \neq \varnothing$) and has the form $\gamma_pF_p[y]$, where $\gamma_p = c_p$ or $-c_p$ as subsequently explained, and $F_p[y]$ is the product of a y component $\Pi(y_k: k \in N[y])$ and an x component $\Pi(x_k: k \in R_p)$, where R_p is defined to be the 'residual set' given by $R_p = N_p - M$. To identify these updates for all relevant indexes p, we define $P_o(M) = \{p \in P: M_p \neq \varnothing\}$ (hence, $P(M) \subset P_o(M)$ by the definition $P(M) = \{p \in P: N_p \subset M\}$).

The sets $N[y]$ vary according to rules that depend on the particular term c_pF_p expanded, generating a term $\gamma_pF_p[y]$ as $N[y]$ ranges over the subsets of M_p. We include reference to the empty subset, which yields $\Pi(y_k: k \in N[y]) = \Pi(y_k: k \in \varnothing) = 1$ (again by the convention that a product over the empty set equals 1). The set R_p can also be empty, which occurs in the case where $N_p = M_p$, giving $R_p = N_p - N_p = \varnothing$ (which arises because N_p itself is a subset of N_p). Then we have $\Pi(x_k: k \in R_p) = 1$ for this case as well.

The stipulation of whether $\gamma_p = c_p$ or $-c_p$ is determined by the number of elements in $N[y]$, so that $\gamma_p = c_p$ if $|N[y]|$ is even and $\gamma_p = -c_p$ if $|N[y]|$ is odd. We identify the different compositions of $N[y]$ and the associated coefficients γ_p by letting $S_h(M_p)$ denote the collection of all h-element subsets of M_p, for $h = 0,1,\ldots,|M_p|$. Then the outcome of flipping the variables x_j for $j \in M$ is identified by specifying the form of $F[y]$ as $N[y]$ ranges over the subsets contained in $S_h(M_p)$ for each $h = 0$ to $|M_p|$. The case for $h = 0$ is the one that gives rise to a set $N[y] = \varnothing$, since there are no 0-element subsets of the set M_p. Thus, for this case we have $S_o(M_p) = \{\varnothing\}$ (the set whose only element is the empty set), and we note that this identity is independent of the set M_p.

Observation 3: Flipping x_j for each $j \in M$ produces the following changes for each set N_p such that $p \in P_o(M)$, and given p, for each $q = 0$ to $|M_p|$ (where $M_p = N_p \cap M$ and $P_o(M) = \{p \in P: M_p \neq \varnothing\}$). Let $\gamma_{pq} = c_p$ if q is even and $\gamma_{pq} = -c_p$ if q is odd and let $\delta_{pq} = \sum(\gamma_{pq}F_{pq}(K): K \in S_q(M_p))$ where $F_{pq}(K) = \Pi(x_k: k \in R_p)\ \Pi(y_k: k \in K)$. Then the change for set N_p is given by

$$\theta_p = \sum\left(\delta_{pq} : q = 0 \text{ to } \left|M_p\right|\right)$$

and the total change for the entire polynomial is given by

$$\sum\left(\theta_p : p \in P_o(M)\right)$$

Proof: The proof of Observation 3 results by iteratively applying the analysis given in the proof of Observation 2. The derivation is straightforward but rather cumbersome and so we omit the details.

The essential elements of Observation 3 may be demonstrated by means of the following example.

3.4 Illustration of Observation 3

Let $M = \{1,2,3\}$, and for simplicity suppose the sets N_p that have non-empty intersections with M are indexed so that $P_o(M) = \{1,2,3,4\}$, where the sets $N_1,\ldots,N_4$ (and their associated sets M_p and R_p, $p = 1,2,3,4$) are given by:

$N_1 = \{1,2,3,4,5\}$	$M_1 = \{1,2,3\}$	$R_1 = \{4,5\}$
$N_2 = \{1,2,4\}$	$M_2 = \{1,2\}$	$R_2 = \{4\}$
$N_3 = \{2,3\}$	$M_3 = \{2,3\}$	$R_3 = \varnothing\ (\Pi\, x_k\colon k \in R_3) = 1)$
$N_4 = \{1\}$	$M_4 = \{1\}$	$R_4 = \varnothing\ (\Pi\, x_k\colon k \in R4) = 1)$

Index sets	*Associated terms* $\gamma_{pq}F_{pq}(K)\colon K \in S_q(M_p)$
For $p = 1$	
$S_o(M_1) = \varnothing$	$c_1x_4x_5(\gamma_{pq} = c_p)$
$S_1(M_1) = \{\{1\},\{2\},\{3\}\}$	$-c_1y_1x_4x_5 - c_1y_2x_4x_5 - c_1y_1x_4x_5\ (\gamma_{pq} = -c_p)$
$S_2(M_1) = \{\{1,2\},\{2,3\},\{1,3\}\}$	$c_1y_1y_2x_4x_5 + c_1y_2y_3x_4x_5 + c_1y_1y_3x_4x_5\ (\gamma_{pq} = c_p)$
$S_3(M_1) = \{\{1,2,3\}\}$	$-c_1y_1y_2y_3x_4x_5(\gamma_{pq} = -c_p)$
For $p = 2$	
$S_o(M_2) = \varnothing$	$c_2x_4(\gamma_{pq} = c_p)$
$S_1(M_2) = \{\{1\},\{2\}\}$	$-c_2y_1x_4 - c_2y_2x_4(\gamma_{pq} = -c_p)$
$S_2(M_2) = \{\{1,2\}\}$	$c_2y_1y_2x_4(\gamma_{pq} = c_p)$
For $p = 3$	
$S_o(M_3) = \varnothing$	$c_3(\gamma_{pq} = c_p)$
$S_1(M_3) = \{\{2\},\{3\}\}$	$-c_3y_2 - c_3y_3(\gamma_{pq} = -c_p)$
$S_2(M_3) = \{\{2,3\}\}$	$c_3y_2y_3(\gamma_{pq} = c_p)$
For $p = 4$	
$S_o(M_4) = \varnothing$	$c_4(\gamma_{pq} = c_p)$
$S_1(M_4) = \{\{1\}\}$	$-c_4y_1(\gamma_{pq} = -c_p)$

As stipulated in Observation 3, these quantities are accumulated and then added to the current coefficients of the corresponding product terms.

Observation 3 allows the update of a multi-flip move to be carried out with greater efficiency than the update of a single flip move, because it avoids updating sets $P(i)$ that may lose or gain elements as a result of coefficients c_p that may change from non-zero to zero or vice versa, when this may occur more than once for the same index p. However, because of the need for expanded data structures to keep track of accumulated terms in such a multi-flip update, we anticipate that it is preferable in most circumstances to

decompose such an update into a series of component updates performed by reference to Observation 2. In our following development, therefore, we focus on an implementation that is organised specifically to exploit such component updates.

We now turn to the question of how to perform these processes efficiently, giving particular emphasis to the situation in which the polynomial is sparse; i.e., where P represents only a small subset of all possible index sets (equivalently, when many terms F_p have associated costs $c_p = 0$ and hence do not explicitly appear in the polynomial).

4 Overall structure of the algorithm

We draw on Observation 2 as the core observation for evaluating and updating moves that transform one solution to another. Consequently, we treat multi-flip moves as composed of a sequence of 1-flip moves rather than seeking to gain efficiencies using Observation 3.

Following such a design, a generic method for the PUB problem may be summarised as follows.

4.1 Generic PUB method

0 Create the initial PUB data structures and select a starting solution. Transform the problem representation so that this solution becomes the current 0 solution.

While a chosen termination condition is not satisfied

1 Select one or more variables x_j to be flipped, by employing the c_j values to identify Δx_o for 1-flip moves as in Corollary 1.2, or the c_p values to identify Δx_o for multi-flip moves as in Corollary 1.2.

2 Apply Observation 2 to update the current problem representation by treating the selected moves as a series of 1-flips. Maintain the appropriate composition of P, and the sets N_p and $P(i)$, for each 1-flip replacing x_j by $1 - x_j$, executed as follows for

$p^* = p[j]$ where $p^* > n$:

If $c_{p^*} = 0$ then

Create the set $N_p{}^*$ and add p^* to the set $P(i)$ for each $i \in N_{p^*}$.

ElseIf $c_{p^*} + c_p = 0$ ($c_{p^*} \neq 0$) then

Remove reference to the set N_{p^*} and drop p^* from P by removing p^* from the set $P(i)$ for each $i \in N_{p^*}$.

Endif

EndWhile

The use of the c_j and c_p values as criteria for selecting a move in Step 1 above may of course be accompanied by additional criteria, such as employing tabu restrictions and aspiration incentives derived from recency and frequency memory in a tabu search method (see, e.g., Glover and Laguna, 1997).

An alternative foundation for describing and justifying the preceding algorithm is given in Appendix 1, which does not rely on employing transformations of variables. We

additionally note that the literature provides a different type of transformation mechanism with the goal of reducing a cubic or higher degree polynomial to a quadratic, and therefore enabling the problem in principle to be solved by a quadratic algorithm. We point out complications that arise by using this quadratic reduction approach by comparison with using our current approach in Appendix 2, causing the quadratic reduction to require multi-flip evaluations to achieve the effect of 1-flip and 2-flip evaluations when using our current approach.

A key challenge for implementing the Generic PUB Algorithm is to specify the manner of structuring the sets $P(i) = \{p \in P: i \in N_p)$ for $i \in N$. As intimated by the algorithm's description, the operations of maintaining the entire problem representation reduce to just the operations of updating the sets $P(i)$, together with maintaining a representation of the sets N_p themselves. In particular, explicit knowledge of the set P is unnecessary, since all relevant knowledge about P is contained in the sets $P(i)$.

In the next section, we show a way to identify the elements of the sets N_p without explicitly listing them, while maintaining a list of c_p values that is dramatically smaller than would be created by allocating a multidimensional matrix to this task (by creating a cost matrix $C(i_1,i_2,\ldots,i_d)$). To accomplish this requires a means to code vectors of the form $(i_1,\ldots,i_h)$ as single numbers v, and to decode such numbers v back into the vectors $(i_1,\ldots,i_h)$ from which they were derived.

5 Coding and decoding index vectors for product terms

We reiterate the convention that the indexes of product terms are written in the form a vector of indexes $(i_1,i_2,\ldots,i_h)$ where $i_1 < i_2 < \ldots\ i_h$ (hence corresponding to the product term $\Pi(x_k: k = i_r: r = 1,\ldots,h)$ where h takes a value between 1 and the degree d of the polynomial). Note that we use the symbol i in this representation because each element i_r identifies an index rather than a variable such as x_k. We first discuss the procedure for coding each such vector of indexes as a single value V. This corresponds to generating the set of indexes p belonging to the set P^α as discussed in Section 3.

5.1 Coding procedure

5.1.1 Organisation

Partition the terms into groups, $G(1)$ to $G(d)$, where each group is a collection of vectors $(i_1,\ldots,i_h)$ for $h = 1,\ldots,d$, identifying the indexes of all possible terms containing h elements (the elements of these vectors constitute the ordered form of the sets N_p that may potentially be created, in a case where the problem is fully dense):

$$G(1): (i_1): i_1 = 1,\ldots,n$$

$$G(2): (i_1,i_2): i_1 = 1,\ldots,n-1;\ i_2 = i_1+1,\ldots,n$$

$$G(3): (i_1,i_2,i_3): i_1 = 1,\ldots,n-2;\ i_2 = i_1+1,\ldots,n-1;\ i_3 = i_2+1,\ldots,n$$

……

$$G(d): (i_1,\ldots,i_d): i_1 = 1,\ldots,n-(d-1);\ i_2 = i_1+1,\ldots,n-d;\ldots;$$
$$i_{d-1} = i_{d-2}+1,\ldots,n-1;\ i_d = i_{d-1}+1,\ldots,n$$

Step 1 Create a base cardinality $\Delta(h)$ and a cumulative cardinality $n(h)$ for each group $G(h)$, $h = 1,\ldots,d$ (the cumulative cardinality is the number of vectors in group G_h added to the number of vectors in all preceding groups).

$$
\begin{aligned}
&G(1):\Delta(1)=n; && n(1)=\Delta(1)\\
&G(2):\Delta(2)=n(n-1)/2; && n(2)=n(1)+\Delta(2)\\
&G(3):\Delta(3)=n(n-1)(n-2)/6; && n(3)=n(2)+\Delta(3)\\
&G(4):\Delta(4)=n(n-1)(n-2)(n-3)/24; && n(4)=n(3)+\Delta(4)\\
&\ldots\ldots\\
&G(d):\Delta(d)=n(n-1)\ldots(n-(d-1))/d! && n(d)=n(d-1)+\Delta(d)
\end{aligned}
$$

Step 2 Create the base coding $v(h)$ for an arbitrary vector $(i_1,\ldots,i_h)$, for each group $G(h)$:

$$
\begin{aligned}
&G(1):v(1)=i_1\\
&G(2):v(2)=(i_2-1)(i_2-2)/2+v(1)\\
&G(3):v(3)=(i_3-1)(i_3-2)(i_3-3)/6+v(2)\\
&G(4):v(4)=(i_4-1)(i_4-2)(i_4-3)(i_4-4)/24+v(3)\\
&\ldots\ldots\\
&G(d):v(d)=\Pi(i_d-q):q=1,\ldots,d)/d!+v(d-1)
\end{aligned}
$$

Step 3 Create the full coding $V(h)$, by adding the cumulative cardinality $n(h-1)$ to $v(h)$ for an arbitrary vector $(i_1,\ldots,i_h)$, for each group $G(h)$:

$$
\begin{aligned}
&G(1):For(i_1):V(1)=v(1)(=i_1)\\
&G(2):For(i_1,i_2):V(2)=n(1)+v(2)\\
&G(3):For(i_1,i_2,i_3):V(3)=n(2)+v(3)\\
&G(4):For(i_1,i_2,i_3,i_4):V(4)=n(3)+v(4)\\
&\ldots\ldots\\
&G(d):For(i_1,\ldots,i_d):V(d)=n(d-1)+v(d)
\end{aligned}
$$

The cumulative cardinality $n(d)$ is the maximum value of $|P|$ for a polynomial of degree d, hence, the maximum number of non-zero coefficients c_p when the polynomial is represented in ascending index order. The coding operation assigns a unique index $p = V[M]$ to each set $M = \{i_1,\ldots,i_h\}$ for $h = 1,\ldots,d$, where $i_1 < \ldots < i_h$. We use the notation $V[M]$ to distinguish the value produced by coding M from the value $V(h)$ that represents the coding value previously defined. Thus, in particular, $V[M] = V(h)$ for the value $V(h)$ calculated by reference to $M = \{i_1,\ldots,i_h\}$ in Step 3 above (because of the ascending index ordering $i_1 < \ldots < i_h$, M may strictly speaking be considered a vector, though we continue to refer to it as a set for convenience). The indexes $p = V[M]$ are exactly those from the set $\{1,2,\ldots,n(d)\}$.

5.2 Using the coding for inputting initial problem data

The coding procedure of Section 5.1 is implemented immediately upon inputting the initial problem data, thereby providing a compact representation to take advantage of situations where the data is sparse.

To handle this as part of the data input procedure is quite simple:

5.2.1 Input procedure

1 Initialise $c_p = 0$ for $p = 1$ to $n(d)$

2 Read problem data and simultaneously generate the coding: let M denote the current set of indexes input from the problem data to become a set N_p, and let c denote the cost associated with M that is to become the value c_p attached to the product term $\Pi(x_k: k \in N_p)$ for $N_p = M$.

3 Produce the index $p = V[M]$ by applying the full coding rule to the elements of M, and let $c_p = c$.

We next identify the operation that is the inverse of the coding operation, and which is a bit more subtle.

5.3 Decoding a coded value p = V to obtain the index set M such that V[M] = V

Let V denote the (full) coded value of an index set $M = \{i_1,\ldots,i_h\}$, where V is the index p of the unknown set $N_p = M$. We seek to identify each component $i_1,\ldots,i_h$ of M (hence of N_p) by performing appropriate operations on the value V.

For an arbitrary real number z, let $[z]$ denote the greatest integer $\leq z$, and let $<z>$ denote the least integer $\geq z$ (hence, when z is a positive non-integer value, then $[z]$ rounds z down and $<z>$ rounds z up).

Then for each value r from 2 to the degree d of the polynomial, we make reference to a constant $\beta(r)$ determined by the following formula

$$\beta(r) = r + 1 - < (r!)1 / r > .$$

The 'rounding up' operator $< >$ is essential to the correctness of this formula. The values $\beta(r)$, for $r = 2,\ldots,d$, may be computed in advance and stored, thus avoiding the need to re-compute these values multiple times when applying the decoding algorithm. For example, if $d = 10$, the relevant $\beta(r)$ values obtained by the preceding formula are as follows: $\beta(2) = 1$, $\beta(3) = 2$, $\beta(4) = 2$, $\beta(5) = 3$, $\beta(6) = 4$, $\beta(7) = 4$, $\beta(8) = 5$, $\beta(9) = 5$, $\beta(10) = 6$ (these values disclose that when $r \leq 5$, $\beta(r)$ can be computed from the simpler formula $c(r) = [(r + 1)/2]$).

We now state the full decoding algorithm.

5.3.1 Decoding algorithm

Step 1 (Identify the group h associated with V.)

The group is $G(1)$ if $V \leq n(1)$; $G(2)$ if $n(1) < V \leq n(2)$; and in general is $G(h)$ if $n(h-1) < V \leq n(h)$.

Step 2 (Convert V to a base code value v)

$v = V - n(h-1)$

Step 3 (Determine the vector $(i_1,i_2,\ldots,i_h)$ from the base code value by generating its components i_r in reverse order, for $r = h, h-1,\ldots,1$.)

$r = h$

---

while $r > 1$
  If $v = 1$ Then
    $ir = r$
  Else
    $w = (v-1)r!$
    $u = [w^{1/r}]$
    $i^* = u + \beta(r)$
      ($i_r = i^*$ or $i^* + 1$, depending on which of these gives the largest value of i_r satisfying
        $\Pi \le w$ for $\Pi = (i_r - 1)(i_r - 2)\ldots(i_r - r)$)
    $\Pi_o = (i^* - 1)(i^* - 2)\ldots(i^* - r + 1)$
    $\Pi_1 = (i^* - r)\,\Pi_o$ (the value of Π if $i_r = i^*$)
    $\Pi_2 = i^*\Pi_o$ (the value of Π if $i_r = i^* + 1$)
    If $\Pi_2 \le w$ then
      $\Pi = \Pi_2$
      $i_r = i^* + 1$
    Else
      $\Pi = \Pi_1$
      $i_r = i^*$
    Endif
    $v := v - \Pi/r!$ (giving the 'residual v' to determine the next component i_r for $r := r - 1$.)
  Endif
$r := r - 1$
EndWhile
$i_1 = v$

We sketch the justification of the decoding method as follows: the validity of Steps 1 and 2 is apparent. The rationale underlying Step 3 derives from the requirement that i_r be the largest integer satisfying

$$\Pi \le w \text{ for } \Pi = (i_r - 1)(i_r - 2)\ldots(i_r - r)$$

This value must clearly be unique and hence, if successfully identified as valid, will uniquely determine each component i_r of the decoded vector. The fact that the iteration over $r = h, h-1,\ldots,1$ actually generates the proper values for Π derives from first observing that setting $i_r = r$ gives i_r the correct value for $v = 1$, and then additionally observing that i_r is appropriately determined for $v = 2$ as a result of the definition of $\beta(r)$

(which applies in particular for $v = 2$ in association with setting $i_r = i^* + 1$). The validity of the determination of i_r for larger values of v can then be established by induction.

The following example illustrates how the coding and decoding processes are implemented.

5.3.2 Illustration of the coding and decoding processes

Assume $d = 4$ and $n = 50$.

5.3.2.1 Coding

We first code the set $M = \{i_1,i_2,i_3\} = \{4,6,11\}$, in order to obtain the index $p = V[M]$ (hence giving the index p of the set $N_p = M$). Applying Step 1, to determine the $n(h)$ values gives:

$$G1 : n(1) = n = 50$$
$$G2 : n(2) = n(1) + n(n-1)/2 = 50 + 1,225 = 1,275$$
$$G3 : n(3) = n(2) + n(n-1)(n-2)/6 = 1,275 + 19,600 = 296,500$$
$$G4 : n(4) = n(3) + n(n-1)(n-2)(n-3)/24 = 296,500 + 230,300 = 526,800$$

Applying Step 2 to give the base coding vector $v(3)$ for $\{4,6,11\}$ we obtain:

$$v(1) = i_1 = 4$$
$$v(2) = v(1) + (i_2 - 1)(i_2 - 2)/2 = 4 + 20/2 = 14$$
$$v(3) = v(2) + (i_3 - 1)(i_3 - 2)(i_3 - 3)/6 = 14 + 720/6 = 134$$

Since $\{i_1,i_2,i_3\}$ is in the group $G(3)$, by Step 3 we obtain the full coding $V(3)$ of $\{4,6,11\}$ from $V(3) = n(2) + v(3) = 1,275 + 134 = 1,409$. The value $V(3) = 1,409$ is therefore the value $p = V[M]$ that is sought.

5.3.2.2 Decoding

For this part of the illustration we decode the full coding value $V = 1298$ to find the set M such that $V[M] = V$. Step 1 of the decoding method checks for membership in one of the groups. Since 1,298 is greater than $n(2) = 1,275$ and not more than $n(3) = 296,500$, we conclude $h = 3$, and hence V represents a vector in the group $G(3)$.

Step 2 generates the basic score $v = V - n(2) = 1,298 - 1,275 = 23$.

We decompose Step 3 into its successive iterations.

- The first iteration of the While loop starts with $v = 23$ and $r = h = 3$.

$$w = (v-1)r! = (23-1)3! = 22 \times 6 = 132$$
$$u = \left\lceil w^{1/r} \right\rceil = [1,32^{1/3}] = 5$$
$$i^* = u + \beta(r) = 5 + \beta(3) = 5 + 2 = 7$$
$$\Pi_o = (i^*-1)(i^*-2)\ldots(i^*-r+1) = (i^*-1)(i^*-2) = 6 \times 5 = 30$$
$$\Pi_1 = (i^*-r)\Pi_o = 4 \times 30 = 120$$
$$\Pi_2 = i^*\Pi_o = 7 \times 30 = 210$$

Since $\Pi_2 > w$ (210 > 132), we have $\Pi = \Pi_1 = 120$ and $i_3 = i^* = 7$.

Finally $v := v - \Pi/r!$ gives $v = 23 - 120/6 = 23 - 20 = 3$.

- The second iteration, for $r = 2$, gives

$$w = (v-1)r! = (3-1)2! = 2 \times 2 = 4$$
$$u = \left\lceil w^{1/r} \right\rceil = \left\lceil 4^{1/2} \right\rceil = 2$$
$$i^* = u + \beta(r) = 2 + \beta(2) = 2 + 1 = 3$$
$$\Pi_o = (i^*-1)(i^*-2)\ldots(i^*-r+1) = (i^*-1) = 2$$
$$\Pi_1 = (i^*-r)\Pi_o = 1 \times 2 = 2$$
$$\Pi_2 = i^*\Pi_o = 3 \times 2 = 6$$

Since $\Pi_2 > w$ (6 > 4), we have $\Pi = \Pi_1 = 2$ and $i^2 = i^* = 3$.

Then $v := v - \Pi/r!$ gives $v = 3 - 2/2 = 2$.

- Hence, on the third iteration, for $r = 1$, we have $i_1 = v = 2$ and the method terminates.

In summary, the index set generated is given by $M = \{i_1,i_2,i_3\} = \{2,3,7\}$ (hence, from the original $V = 1{,}298$, we have $N_{1,298} = \{2,3,7\}$).

5.4 Illustration of memory required for the cost values c_p and the associated sets N_p

The amount of memory required to store the costs c_p and the associated sets N_p using the foregoing processes is equal to the value $n(d)$ for a polynomial of degree d when using an ascending index order representation. For purposes of illustration, we calculate these values for the cases $d = 1$ to 5, assuming $n = 100$ in each case. For comparison, we show the amount of memory required by using a cost matrix of the form $C(i_1,\ldots,i_d)$, which entails a storage space of n_d.

Values of d	*1*	*2*	*3*	*4*	*5*
Cost matrix memory n_d	100	10,000	1,000,000	100,000,000	10,000,000,000
Coded memory $n(d)$	100	5,050	166,750	4,087,957	79,375,495

The rapid growth in the size of memory depicted by this illustration suggests that a polynomial of degree 4 or 5 may be the largest that is practical to work with. However, we note that this memory does not take account of the fact that the problem will generally be sparse, so that only a few percent of the total number of possible product terms may actually exist at any given time in the problem formulation. We address the challenges of taking advantage of our results within the context of exploiting sparsity, which is a hallmark of real world problems, in Part 2 of this paper.

6 Conclusions

We have provided basic relationships for developing efficient algorithms to solve PUB problems, and have identified coding and decoding processes that give the raw materials

for creating special algorithms to exploit practical problems that contain sparse matries. The sequel to this work in Part 2 will give special types of memory structures and associated updating algorithms to take advantage of the foundations laid in Part 1, to yield significant improvements both in memory consumed and in the speed of executing an algorithm for the PUB problem.

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.

Boros, E. and Hammer, P.L. (2002) 'Pseudo-Boolean optimization', *Discrete Applied Mathematics*, Vol. 123, Nos. 1–3, pp.155–225.

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. and Hao, J.K. (2010a) 'Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems', *International Journal of Metaheuristics*, Vol. 1, No. 1, pp.3–10.

Glover, F. and Hao, J.K. (2010b) 'Fast 2-flip move evaluations for binary unconstrained quadratic optimisation problems', *International Journal of Metaheuristics*, Vol. 1, No. 2, pp.100–107.

Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer Academic Publishers.

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

Hanafi, S., Rebai, A-R. and Vasquez, M. (2010) 'Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems', Working paper, LAMIH, Université de Valenciennes.

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

Kochenberger, G. (2010) 'Notes on 3-SAT and max 3-SAT', Working paper, University of Colorado, Denver.

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

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.

Notes

1 The indexing selected for this illustration is based on supposing that $n < 30$, to support the convention that $N_p = \{p\}$ for $p \le n$ (as illustrated by $N_3 = \{3\}$). We have taken the liberty of choosing the indexes $p > n$ for the preceding sets N_p by making them somewhat smaller than they would likely be under ordinary circumstances.

Appendix 1

Alternative foundation for the generic PUB method

The main results that support the generic PUB method can be based on a different foundation, which does not rely on creating a problem transformation to recast each iteration as starting from the solution $x = 0$. To express this, we introduce cross product terms $F_p(i)$ and values $v(i)$ associated with the sets $P(i) = \{p \in P: i \in N_p\}$ for PUB as follows:

$$F_p(i) = \Pi\left(x_k : k \in N_p - \{i\}\right)$$

$$v(i) = \sum\left(c_p F_p(i) : p \in P(i)\right)$$

As before, we begin our analysis from the 1-flip perspective, and let x' and x'' represent two binary solutions where x'' is obtained from x' by flipping the value of a single variable x_i from 0 to 1 or from 1 to 0. Similarly, we define $x_o' = \sum\left(c_p F_p' : p \in P\right)$, where $F_p' = \Pi\left(x_p' : i \in N_p\right)$ and $x_o'' = \sum\left(c_p F_p : p \in P\right)$, where $F_p'' = \Pi\left(x_i'' : i \in N_p\right)$, and define $\Delta x_o' = x_o'' - x_o'$.

Let $F_p'(i)$ and $v'(i)$ and be the instances of $F_p(i)$ and $v(i)$ that result for $x = x'$ and let $F_p''(i)$ and $v''(i)$ be the corresponding instances that result for $x = x''$, hence, $F_p'(i) = \Pi\left(x_k' : k \in N_p - \{i\}\right)$, $v'(i) = \sum\left(c_p F_p'(i) : p \in P(i)\right)$, etc.

Proposition 1: $\Delta x_o' = \left(1 - 2x_i'\right)v'(i),\ i \in M$.

Proof: Write x_o in the form

$$x_o = \sum\left(c_p F_p : p \in P(i)\right) + \sum\left(c_p F_p : p \in P - P(i)\right)$$

Given that the sets N_p for $p \in P(i)$ are precisely those that contain the index i, the foregoing may be re-written as

$$x_o = x_i \sum\left(c_p F_p(i) : p \in P(i)\right) + \sum\left(c_p F_p : p \in P - P(i)\right)$$

The value $\Delta x_o' = x_o'' - x_o'$ therefore can be written

$$\Delta x_o' = x_o'' - x_o' = x_i'' \sum\left(c_p F_p''(i) : p \in P(i)\right) - x_i' \sum\left(c_p F_p'(i) : p \in P(i)\right)$$

since $\sum\left(c_p F_p^{'} : p \in P - P(i)\right) = \sum\left(c_p F_p^{''} : p \in P - P(i)\right)$ as a consequence of the fact that x_i is not a part of any F_p for $p \in P - P(i)$. Moreover, since x_i is not represented in either $F_p^{''}(i)$ or $F_p^{'}(i)$ for $p \in P(i)$, and x_i is the only variable that changes its value, we have $F_p^{''}(i) = F_p^{'}(i)$ for $p \in P(i)$. Hence,

$$\Delta x_o^{'} = \left(x_i^{''} - x_i^{'}\right)\sum\left(c_p F_p^{'}(i) : p \in P(i)\right)$$

Finally from $v'(i) = \sum\left(c_p F_p^{'}(i) : p \in P(i)\right)$ we obtain $\Delta x_o^{'} = \left(1 - 2x_i^{'}\right)v'(i)$, $i \in M$, as stipulated.

Proposition 1, which effectively constitutes an alternative formulation of Corollary 2 in Section 2, directly generalises the corresponding result of Glover and Hao (2010a) for quadratic problems, and likewise shows that the amount of effort to compute $\Delta x_o^{'}$ is the same for general polynomial objective functions as for quadratic objective functions. Thus, in particular, by this representation, when the problem data is first set up, and we have a starting solution x', we compute and save the values $v'(i) = \sum\left(c_p F_p^{'}(i) : p \in P(i)\right)$ for each $i \in M$. The critical element for exploiting this proposition in the most effective manner is to identify a way to update the $v'(i)$ values efficiently from one iteration to the next, as x' is updated to become the solution previously denoted by x''.

Next define

$$P(i:j) = \left\{p \in P(i) : j \in N_p\right\} = \left\{p \in P : i, j \in N_p\right\} = P(i) \cap P(j)$$
$$F_p(i:j) = \Pi\left(x_k : k \in N_p - \{i, j\} \text{ for } p \in P(i:j)\right)$$
$$v(i:j) = \sum\left(c_p F_p(i:j) : p \in P(i:j)\right)$$

We are interested in identifying the new value $v''(i)$ that replaces $v'(i)$ when x'' results from x' by $x_j^{''} = 1 - x_j^{'}$.

Proposition 2: $v''(i) = v'(i) + \left(1 - 2x_j^{'}\right)v'(i:j)$

Proof: From the definition $v(i) = \sum\left(c_p F_p(i) : p \in P(i)\right)$, we may break the summation into two components to give

$$v(i) = \sum\left(c_p F_p(i) : p \in P(i:j)\right) + \sum\left(c_p F_p(i) : p \in P(i) - P(i:j)\right)$$

Since the sets N_p for $p \in P(i: j)$ are precisely those for $p \in P(i)$ that include the index j, the summation on the left may be rewritten as $x_j \sum\left(c_p F_p\left(i:j\right) : p \in P\left(i:j\right)\right)$. By the same token, none of the components of the summation on the right will change when replacing x' by x'', hence the value $\Delta v'(i) = v''(i) - v'(i)$ will be given by

$$\Delta v'(i) = x_j'' \sum \left(c_p F_p''(i:j) : p \in P(i:j) \right) - x_j' \sum \left(c_p F_p'(i:j) : p \in P(i:j) \right).$$

However, by definition, $F_p(i: j)$ excludes reference to the index j, and hence $F_p''(i:j) = F_p'(i:j)$. Consequently, we obtain:

$$\Delta v'(i) = \left(x_j'' - x_j' \right) \sum \left(c_p F_p'(i:j) : p \in P(i:j) \right).$$

and finally by the substitution $1 - x_j'$ for x_j'' and $v'(i:j)$ for

$$\sum \left(c_p F_p'(i:j) : p \in P(i:j) \right) \text{ gives } \Delta v'(i) = \left(1 - 2x_j' \right) v'(i:j)$$

which yields the value $v''(i) = v'(i) + \left(1 - 2x_j' \right) v'(i:j)$ for $v''(i)$ stated in the proposition.

Proposition 2 likewise generalises the corresponding result of Glover and Hao (2010a) for quadratic problems. By this Proposition 2, we identify the value $v'(i:j)$ and then obtain the updated $v''(i)$ value that becomes $v'(i)$ on the next iteration. If $v'(i:j)$ itself has been maintained in updated form, this likewise is a $O(1)$ operation.

In general, consider a sequence of terms $P(i:j:k:\ldots)$, $F_p(i,j,k:\ldots)$ and $v(i:j:k:\ldots)$, which go as far as necessary to carry out the updates needed at each level. The structure for doing this is as follows.

Let I be a subset of N that does not contain index j and let $J = I \cup \{j\}$. Then define

$$P(J) = \left\{ p \in P : J \subset N_p \right\} = P(I) \cap P(j)$$
$$F_p(J) = \Pi \left(x_k : k \in N_p - J \right) \text{for } p \in P(J)$$
$$v(J) = \sum \left(c_p F_p(J) : p \in P(J) \right)$$

(Note $P(\varnothing) = P$, $F_p(\varnothing) = F_p$ and $v(\varnothing) = v$) If J contains a single element, $J = \{j\}$, then $I = \varnothing$, and we for simplicity we denote $P(J)$ by $P(j)$, hence, yielding $P(j) = \{p \in \text{P}: j \in N_p\}$, $F_p(j) = \Pi(x_k: k \in N_p - \{j\})$, $v(j) = \sum \left(c_p F_p(j) : p \in P(j) \right)$, corresponding to our earlier definition.

We want to obtain $v''(i)$ for all $i \in N_o$ after flipping x_j. Consider first the sets J that are the maximal sets N_p containing j. Let $P_{\max}(j) = \{p \in P: j \in N_p$ and N_p is maximal (no set N_q strictly contains N_p for $q \in P)\}$.

Lemma: If J is a maximal set N_p containing j, i.e., $p^* \in P_{\max}(j)$, and $J = N_p{}^*$, then $v(J) = c_{p^*}$ for all x (hence, in particular, $v'(J) = c_{p^*}$).

Proof: $J = N_{p^*}$ where $p^* \in P_{\max}(j)$ implies $P(J) = \{p^*\}$, since N_{p^*} is the unique set containing J. In turn, the definition $F_p(J) = \Pi(x_k: k \in N_p - J)$ implies $F_p(J) = 1$, since J is maximal and hence $N_p - J = \varnothing$. Finally, by the definition $v(J) = \sum \left(c_p F_p(J) : p \in P(J) \right)$ we have $v(J) = c_{p^*} F_{p^*}(J)$, which gives $v(J) = c_{p^*}$.

Now we obtain the following.

General proposition: $v''(I) = v'(I) + \left(1 - 2x_j'\right) v'(J)$

Proof: From the definition $v(I) = \sum \left(c_p F_p(I) : p \in P(I) \right)$, we may break the summation into two components to give

$$v(I) = \sum \left(c_p F_p(I) : p \in P(J) \right) + \sum \left(c_p F_p(I) : p \in P(I) - P(J) \right)$$

Since the sets N_p for $p \in P(J)$ are precisely those for $p \in P(I)$ that include the index j, the summation on the left may be rewritten as $x_j \sum \left(c_p F_p(J) : p \in P(J) \right)$. By the same token, none of the components of the summation on the right will change when replacing x' by x'', hence the value $\Delta v'(I) = v''(I) - v'(I)$ will be given by

$$\Delta v'(I) = x_j'' \sum \left(c_p F_p''(J) : p \in P(J) \right) - x_j' \sum \left(c_p F_p'(J) : p \in P(J) \right).$$

However, by definition, $F_p(J)$ excludes reference to the index j, and hence $F_p''(J) = F_p'(J)$. Consequently, the expression for $\Delta v'(I)$ reduces to

$$\Delta v'(I) = \left(x_j'' - x_j' \right) \sum \left(c_p F_p'(J) : p \in P(J) \right).$$

and finally by the substitution $1 - x_j'$ for x_j'' and $v'(J)$ for $\sum \left(c_p F_p'(J) : p \in P(J) \right)$ gives

$$\Delta v'(I) = \left(1 - 2x_j'\right) v'(J)$$

and hence

$$v''(I) = v'(I) + \left(1 - 2x_j'\right) v'(J) \text{ for } v''(I)$$

as stated in the proposition.

The methods of this paper and of the Part 2 sequel can likewise use the preceding results as a starting point.

Appendix 2

Transformation for reducing a higher degree polynomial to a quadratic

Boros and Hammer (2002) provide a penalty transformation approach that can be applied iteratively to reduce a higher degree polynomial to a quadratic. The mechanism can be depicted by considering the case where we seek to replace a cubic formulation by a quadratic formulation. The rules of the approach for this case are as follows.

a identify a cubic term $x_i x_j x_k$ for reduction

b choose a two variable product term within the cubic term, say $x_i x_j$, which will be replaced by a new binary variable x_{ij} (Every cubic term $x_i x_j x_h$ that contains the product $x_i x_j$ will become replaced by the associated term $x_{ij} x_k$. Of course, the

variables may appear in a different order in the original cubic terms, as for example $x_j x_h x_i$ or $x_h x_j x_i$.)

c add a quadratic penalty term $P(x_i x_j - 2x_i x_{ij} - 2x_j x_{ij} + 3x_{ij})$

By means of this transformation, the penalty impact on the objective function is zero when $x_{ij} = x_i x_j$ and is P otherwise.

We now analyse the effect of this approach in relation to using the PUB approach of the present paper. Consider the use of 1-flip moves with the quadratic reduction procedure in the situation where the product $x_i x_j$ introduces a variable x_{ij} by the transformation indicated above. As long as both x_i and x_j equal 0 there is no problem. However, a complication ensues when at least one of x_i and x_j equals 1. First suppose, for instance that $x_i = 1$ and $x_j = 0$ (together with $x_{ij} = 0$, as appropriate). Upon attempting to evaluate the 1-flip $x_j = 0 \rightarrow 1$, we incur a penalty of P. The same thing happens if we instead evaluate the 1-flip $x_{ij} = 0 \rightarrow 1$. Thus, to accurately evaluate the move $x_j = 0 \rightarrow 1$ we must in fact evaluate the 2-flip move that simultaneously executes $x_j = 0 \rightarrow 1$ and $x_{ij} = 0 \rightarrow 1$.

Next, suppose both x_i and x_j equal 1 (together with $x_{ij} = 1$, as appropriate). If we now evaluate either the 1-flip $x_i = 1 \rightarrow 0$ or the 1-flip $x_j = 1 \rightarrow 0$, we again incur a penalty of P (and the same thing happens if we instead evaluate 1-flip $x_{ij} = 1 \rightarrow 0$). In this situation, it is again necessary to evaluate a 2-flip move to accurately identify the outcome of the considered change, hence either simultaneously executing $x_i = 1 \rightarrow 0$ and $x_{ij} = 1 \rightarrow 0$ or executing $x_j = 1 \rightarrow 0$ and $x_{ij} = 1 \rightarrow 0$.

The complications do not stop here, however. Suppose that x_j is contained in two product terms $x_i x_j$ and $x_h x_j$ that have undergone a transformation, producing the variables x_{ij} and x_{hj}. In addition to the need to evaluate 2-flip moves in the situation described above (which applies to x_h and x_j as well as to x_i and x_j), an additional difficulty is encountered if both $x_i = 1$ and $x_h = 1$. Then for the case where $x_j = 0$, to accurately evaluate the 1-flip $x_j = 0 \rightarrow 1$ requires evaluating the 3-flip $x_j = 0 \rightarrow 1$, $x_{ij} = 0 \rightarrow 1$, $x_{hj} = 0 \rightarrow 1$. On the other hand, for the case where $x_j = 1$, to accurately evaluate the 1-flip $x_j = 1 \rightarrow 0$ requires evaluating the 3-flip $x_j = 1 \rightarrow 0$, $x_{ij} = 1 \rightarrow 0$, $x_{hj} = 1 \rightarrow 0$.

Situations can similarly arise requiring the evaluation of still higher-order flip moves in order to determine an accurate evaluation of changing the value of a single variable. By contrast, the PUB approach we propose permits all of these situations to be handled by only evaluating 1-flip moves. If we allow the use of 2-flip moves with the transformation approach and our PUB approach, in the hope of thereby reducing the number of problematical situations encountered by the transformation approach, we discover instead that the number of these situations increases. For example, a 2-flip that implicates two variables x_{ij} and x_{pq} can require four flips to evaluate accurately in the transformation approach, and interrelated product terms can produce even greater complications, all of which are handled directly by a 2-flip in the PUB approach.

When the Boros and Hammer transformation is applied iteratively to reduce polynomials of degree greater than 3 to quadratics, each reduction in the degree d compounds the effects illustrated for reducing d from 3 to 2. The result therefore incurs complex combinations of penalties that produce inaccurate evaluations of 1-flip and 2-flip moves that can only be rectified by evaluating moves that flip additional variables.