Noname manuscript No.
(will be inserted by the editor)

Backbone Guided Tabu Search for Solving the UBQP Problem

Yang Wang · Zhipeng Lü · Fred Glover · Jin-Kao Hao*

Received: date / Accepted: date

Abstract We propose a backbone-guided tabu search (BGTS) algorithm for the Unconstrained Binary Quadratic Programming (UBQP) problem that alternates between two phases: (1) a basic tabu search procedure to optimize the objective function as far as possible; (2) a strategy using the TS notion of strongly determined variables to alternately fix and free backbone components of the solutions which are estimated likely to share values in common with an optimal solution. Experimental results show that the proposed method is capable of finding the best-known solutions for 21 large random instances with 3000 to 7000 variables and boosts the performance of the basic TS in terms of both solution quality and computational efficiency.

keywords: Backbone-Guided Search, Tabu Search, UBQP, Strongly Determined Variables, Variable Fixing and Freeing.

Yang Wang
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: yangw@info.univ-angers.fr

Zhipeng Lü
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: lu@info.univ-angers.fr, zhipeng.lui@gmail.com

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

Jin-Kao Hao (Corresponding author)
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: hao@info.univ-angers.fr
http://www.info.univ-angers.fr/pub/hao/

1 Introduction

The unconstrained binary quadratic programming problem, denoted by UBQP, can be formulated as follows:

$$f(x) = max \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j \tag{1}$$

where $[q_{ij}]$ is an a n by n matrix of constants and x is an n-vector of binary (zero-one) variables, i.e., $x_i \in \{0, 1\},\ i = 1, \ldots, n$.

In addition to its theoretical significance as a canonical NP-hard problem (Garey and Johnson (1979)), the UBQP is notable for its ability to formulate a wide range of important problems, including those from computer aided design (Krarup and Pruzan (1978)), social psychology (Harary (1953)), financial analysis (McBride and Yormark (1980)), machine scheduling (Alidaee et al (1994)), cellular radio channel allocation (Chardaire and Sutter (1994)), the vertex coloring problem (Kochenberger et al (2005)), the set packing problem (Alidaee et al (2008)), the set-partitioning problem (Lewis et al (2008)) and the linear ordering problem (Lewis et al (2009)).

The application potential of UBQP is much greater than might be imagined, due to the ability to incorporate quadratic infeasibility constraints into the objective function in an explicit manner. This re-formulation process enables UBQP to serve as a common model for a wide range of combinatorial optimization problems. A review of additional applications and the reformulation procedures can be found in Kochenberger et al (2004) demonstrating the utility of UBQP for a variety of applications.

Considering the relevance of the UBQP across a broad spectrum of problems, a number of exact algorithms have been proposed in the literature. The typical approaches include those of Boros et al (2008); Horst et al (2000), using branch and bound or branch and cut. However, due to the high computational complexity of UBQP, a variety of problems arising from practical applications, except those problems of sizes less than $n = 100$, have proved to be intractable for these exact approaches.

For larger instances, exact methods become prohibitively expensive to apply. By contrast, variants of metaheuristic algorithms have been extensively studied to solve UBQP and shown to be effective to find high-quality solutions in a reasonable time. Some representative metaheuristic methods include local search (Boros et al (2007)); Simulated Annealing (Katayama and Narihisa (2001)); adaptive memory approaches based on Tabu Search (Glover et al (1998, 2010); Palubeckis (2004, 2006)); population-based approaches such as Evolutionary Algorithms (Borgulya (2005)), Scatter Search (Amini et al (1999)) and Memetic Algorithms (Lü et al (2010); Merz and Katayama (2004)); and specially designed one-pass heuristics (Glover et al (2002)).

Among these procedures, Tabu Search (TS) represents one of the most successful approaches. One of the first adaptive memory TS algorithms for the UBQP (Glover et al (1998)), for instance, has been used to solve applications

arising in a wide variety of settings. Recently, several restart TS strategies have been explored in Palubeckis (2004) which obtained good results on large problem instances. A sequel further improves these results by an iterated tabu search algorithm (Palubeckis (2006)). Furthermore, a recent diversification-driven tabu search method (Glover et al (2010)) has been demonstrated to be effective on a wide range of random UBQP problem instances.

The remaining part of the paper is organized as follows. In Section 2, we present the ingredients of our backbone-guided tabu search (BGTS) algorithm which includes a basic tabu search procedure and associated mechanisms for the fixing and freeing variables. Section 3 is dedicated to computational results. Section 4 investigates several essential components of the BGTS algorithm and concluding remarks are given in Section 5.

2 Backbone-Guided Tabu Search for UBQP

The *backbone* terminology comes from the context of the well-known satisfiability problem (SAT). Informally, the backbone of a SAT instance is the set of literals [1] which are true in every satisfying truth assignment (Monasson et al (1998); Kilby et al (2005)). Zhang (2004) presents an effective backbone-based heuristic for SAT and an example of a similar strategy is reported for the multi-dimensional knapsack problem in Wilbaut et al (2009). Such a strategy was also proposed in connection with exploiting *strongly determined* and *consistent* variables in Glover (1977), and has come to be one of the basic strategies associated with tabu search. (Discussions of this strategy in multiple contexts appear in Glover and Laguna (1997) and in Glover (2005).)

We restrict attention in this paper to the "strongly determined" aspect of strongly determined and consistent variables, and borrow the "backbone" terminology from the SAT literature as a vehicle for naming our procedure. The SAT notion of a backbone refers to a set of variable assignments that are shared by all the global optima of an instance. From a practical standpoint this definition clearly has little utility since we do not know these global optima in advance and our goal is to find one of them. Hence we instead take an approach based on available knowledge by keeping track of one or more solutions generated during the course of the search that exhibit the highest quality, and use the criterion of being strongly determined as an indicator of those assignments that likely to be shared in common with a global optimum. In particular, we use a simplification of the notion of Glover (1977) by considering a variable to be strongly determined if changing its assigned value in a high quality solution will cause the quality of that solution to deteriorate significantly. Identifying a backbone according to this criterion, we then "instantiate" the backbone by fixing the values of those variables that qualify for membership.

[1] A literal is a boolean variable or the negation of a boolean variable.

2.1 Main Scheme and General Idea

Algorithm 1 describes the main procedure of our BGTS algorithm. Starting from a random initial solution x^S, a basic TS procedure (See Section 2.2) is executed to reach a local optimum (line 15) and one or more of the best solutions obtained by TS during its run are recorded as the reference solution(s) which are used for backbone identification (line 16). The algorithm decides then to fix or free variables according to whether the best solution obtained in the current run of TS is better than that of the last round (f_p). If this is the case, a variable fixing phase is launched where some variables are selected to be fixed as backbone variables (lines 21-28). Otherwise, a freeing phase is triggered to release some fixed backbone variables so that their values can be changed (lines 30-36).

Once a set of variables is selected to be fixed or freed, a new round of TS begins with the non-backbone variables randomly assigned. We employ the original design for exploiting strongly determined variables by keeping the variables fixed after passing a solution to the TS procedure, by compelling the assigned values to remain fixed once the TS method is launched. (An option would allow the fixed variables eventually to be freed within the TS procedure after a chosen period. We include this option as one of the possibilities to be examined in future studies.) If the TS method finds an improved solution as a result of starting from a given backbone, then the size of the backbone is increased, thus diminishing the number of variables that receive random assignments the next time the TS method is executed. In reverse, if the TS method fails to find an improved solution when starting from a given backbone, then the backbone is reduced, thus increasing the number of variables that receive random assignments on the next pass.

Therefore, the BGTS algorithm repeatedly alternates between a phase of tabu search and a phase that either fixes or frees selected variables until a stop criterion is satisfied. We call each round of these two phases a *trial*.

It should be noted that when $f(x') > f_p$ is not satisfied in Algorithm 1, it is possible that the backbone B is empty ($nb = 0$), and in this case the set $B(-)$ will be empty and the method simply launches another iteration with an empty backbone ($nd = nb = 0$). However, if $f(x') > f_p$, our algorithm guarantees that the set of freeing variables F is not empty ($nf > 0$) since we employ a geometric ratio strategy to set the appropriate number of variables to be fixed as backbones at each iteration (see Section 2.4.2).

2.2 TS Procedure

Our version of TS used in this work is a very simple tabu search procedure implemented in Glover et al (2010). The neighborhood of a given configuration is obtained by flipping the value of a single variable x_i to its complementary value $1 - x_i$.

Algorithm 1 Pseudo-code of the BGTS algorithm for UBQP

1: **Input**: matrix Q
2: **Output**: the best binary n-vector x^* found so far
3: // Tabu Search Procedure: line 15
4: // Reference Solution Construction: line 16
5: // Fixing Phase: lines 21-28
6: // Freeing Phase: lines 30-36
7: Initialization: $B = \{\}$, $F = \{x_1, \ldots, x_n\}$, $f(x^*) = 0$, $f_p = 0$
8: // B = index set for the Backbone variables, $nb = |B|$
9: // F = index set for the Free variables, $nf = |F|$
10: // *Comment*: $B \cup F = N$ ($N = \{1, \ldots, n\}$), $B \cap F = \emptyset$ and $nb + nf = n$
11: **repeat**
12: Let x^S denote the starting solution for each iteration of the algorithm
13: Let x_i^B, for $i \in B$, denote the current values assigned to the backbone variables
14: $x_i^S = x_i^B$ for $i \in B$, and $x_i^S = Rand[0, 1]$ for $i \in F$
15: $x' \leftarrow$ Tabu_Search(x^S) /* Section 2.2 */
16: Obtain the reference solution(s) P from current TS procedure /* Section 2.3.2 */
17: **if** $f(x') > f(x^*)$ **then**
18: $x^* = x'$
19: **end if**
20: **if** $f(x') > f_p$ **then**
21: // **Rule for Fixing Variables:**
22: Order the elements $i \in F$ so that $score(i_1) \leq \ldots \leq score(i_{nf})$ /* Section 2.4.1 */
23: Let $B(+) = \{i_1, \ldots, i_{na}\}$ ($na \leq nf$) /* Section 2.4.2 */
24: // na = the number of variables to be added to the backbone when variables are fixed
25: $B := B \cup B(+)$ ($nb := nb + na$)
26: $F := F - B(+)$ ($nf := nf - na$)
27: $x_i^B = x_i'$ for $i \in B(+)$ (x_i^B is already determined for $i \in$ "previous B"$:= B - B(+)$)
28: Fix the variables in $B(+)$ as *backbone* variables /* Section 2.4.2 */
29: **else**
30: // **Rule for Freeing Variables:**
31: Order the elements $i \in B$ so that $score(i_1) \geq \ldots \geq score(i_{nb})$ /* Section 2.5.1 */
32: Let $B(-) = \{i_1, \ldots, i_{nd}\}$ ($nd \leq nb$) /* Section 2.5.2 */
33: // nd = the number of variables to be dropped from the backbone in variable freeing
34: $B := B - B(-)$ ($nb := nb - nd$)
35: $F := F \cup B(-)$ ($nf := nf + nd$)
36: Free the variables in $B(-)$ as *non-backbone* variables /* Section 2.5.2 */
37: **end if**
38: $f_p = f(x')$
39: **until** a stop criterion is met

For large problem instances, it is necessary to be able to rapidly determine the effect of a move on the objective function $f(x)$. In our implementation, this neighborhood uses a fast incremental evaluation technique introduced in Glover et al (1998) and enhanced in Glover and Hao (2010) to calculate the cost (move value) of transitioning to each neighboring solution. The procedure maintains a data structure that stores the move value (change in $f(x)$) for each possible move, and employs a streamlined calculation for updating this data structure after each iteration. Specifically, once a move is performed, one only needs to update a subset of move values affected by the move.

Our basic tabu search incorporates a *tabu list* as a "recency-based" memory structure to assure that solutions visited within a certain span of iterations,

called the tabu tenure, will not be revisited (Glover and Laguna (1997)). In our implementation, we elected to set

$$TabuTenure(i) = tl + rand(10) \tag{2}$$

where tl is a given constant and rand(10) takes a random value from 1 to 10.

The TS algorithm then restricts consideration to variables which are not forbidden by the tabu list, and selects a variable to flip that produces the best (largest) incremental objective value, breaking ties randomly.

Together with this rule, a simple aspiration criterion is applied which allows a move to be performed in spite of being tabu if it leads to a solution better than the current best solution. Our TS procedure stops when the best solution cannot be improved within a given number α of moves and we call this number the *improvement cutoff*.

2.3 Basic Preliminaries

In this section we give some basic definitions used in our BGTS algorithm.

2.3.1 Contribution of a Variable

Definition 1. Relative to a given solution $x' = \{x'_1, x'_2, ..., x'_n\}$ and a specific variable x_i, $i \in N$, let $\sigma = 1$ if $x'_i = 0$ and let $\sigma = -1$ if $x'_i = 1$. Then the (objective function) *contribution* of x_i in relation to x' is defined as:

$$VC_i(x') = \sigma(q_{ii} + \sum_{j \in N \setminus \{i\}} q_{ij} x_j) \tag{3}$$

As noted in Glover et al (1998) and in a more general context in Glover and Hao (2010), $VC_i(x')$ identifies the change in $f(x)$ that results from changing the value of x'_i to 1 - x'_i; i.e.,

$$VC_i(x') = f(x'') - f(x') \tag{4}$$

where $x''_j = x'_j$ for $j \in N - \{i\}$ and $x''_i = 1 - x'_i$. We observe that under a maximization objective if x' is a locally optimal solution, as will typically be the case where we select x' to be a high quality solution, then $VC_i(x') \leq 0$ for all $i \in N$, and the current assignment $x_i = x'_i$ will be more strongly determined as $VC_i(x')$ is "more negative".

Definition 2. Relative to a given population of solutions $X(P) = \{x^1, \ldots, x^p\}$ indexed by $P = \{1, \ldots, p\}$, and relative to a chosen variable x_i, let $P_i(0) = \{k \in P : x^k_i = 0\}$ and $P_i(1) = \{k \in P : x^k_i = 1\}$. Then the (objective function) *contribution* of x_i in relation to P is defined as follows.

Contribution for $x_i = 0$:

$$VC_i(P:0) = \sum_{k \in P_i(0)} VC_i(x^k) \tag{5}$$

Contribution for $x_i = 1$:

$$VC_i(P:1) = \sum_{k \in P_i(1)} VC_i(x^k) \tag{6}$$

We remark that Definition 2 is a straightforward generalization of Definition 1 from the following perspective. Consider the situation where the population $X(P) = \{x'\}$; i.e., $x' = x^1$ and $P = \{1\}$. Then if $x'_i = 0$ we have $VC_i(P:0) = VC_i(x')$ and $VC_i(P:1) = 0$, while if $x'_i = 1$ we have $VC_i(P:1) = VC_i(x')$ and $VC_i(P:0) = 0$ (where a summation over the empty set is understood to be 0). The cases for setting $x_i = 0$ when $x'_i = 0$ and for setting $x_i = 1$ when $x'_i = 1$ cause no change in $f(x)$ (i.e., $f(x'') - f(x') = 0$ when $x'' = x'$). Hence the stipulations $VC_i(P:1) = 0$ when $x'_i = 0$ and $VC_i(P:0) = 0$ when $x'_i = 1$ may be taken as implicit (though irrelevant) in Definition 1.

2.3.2 Reference Solution

During a run of TS, one or more best solutions are collected which are used as reference solutions for the purpose of fixing or freeing variables. In our BGTS algorithm, we obtain the reference solutions in two ways:

- **Single Solution:** Take a single best solution obtained by the current round of TS as the reference solution.
- **Solution Population:** Take a given number p of the best but different solutions from the current round of TS, which then constitute a solution population. In our implementation, we take $p = 20$ as indicated in Table 1.

2.3.3 Two BGTS Variants

Our two key variants of BGTS consist of either using $VC_i(x')$ (Single Solution) or using $VC_i(P:0)$ and $VC_i(P:1)$ (Solution Population) to evaluate the effect of changing the value of x_i from x'_i to $1 - x'_i$ or, in general, the effect of setting $x_i = 0$ or 1. These two variants are respectively denoted by SS (for Single Solution) and SP (for Solution Population).

2.4 Fixing Procedure

Given the reference solution(s), our fixing procedure operates according to three steps:

1. *Scoring*: calculate a score for each non-backbone variable;
2. *Selecting*: choose a certain number of non-backbone variables;
3. *Fixing*: fix the assignments for the chosen variables so that these variables are compelled to receive their indicated values upon launching the next round of TS.

2.4.1 Non-Backbone Variable Scoring

We use a variable's contribution value (defined in Section 2.3.1) to score all the non-backbone variables, among which we select a certain number of variables as backbone variables to be fixed. The main idea is that the smaller the contribution $VC_i(x')$ made by variable x_i according to Definition 1, then the greater will be the amount of deterioration in $f(x)$ caused by changing x_i's value from $x_i = x'_i$ to $x_i = 1 - x'_i$, and hence the more strongly determined the assignment $x_i = x'_i$ will be. Similarly, the smaller the value of $VC_i(P : v)$ for $v = 0$ or 1 in Definition 2, the more strongly determined we may consider the assignment $x_i = v$ to be. In this case $v = x'_i$, referring to the 0 or 1 value taken by x_i in a specified solution x'. We normally understand x' to correspond to one of the solutions x^k in the population $X(P)$.

Rule 1. $score(i) = VC_i(x')$

Rule 2. $score(i) = VC_i(P : x'_i)$

2.4.2 Backbone Variable Selection and Fixing

When all the non-backbone variables are scored, we sort them according to their scores in a non-decreasing order. Then, we decide how many non-backbone variables with the smallest contribution scores should be fixed at their associated values x'_i to become backbone variables at the current trial.

Based on the fact that in the initial stage, the number of non-backbone variables is large while this number becomes smaller and smaller through a series of passes when the TS method finds progressively improved solutions, we employ a strategy that fixes a gradually reduced number of backbone variables throughout such a succession of improvements. Specifically, the number of backbone variables at the first fixing phase is relatively large and is then gradually reduced with a geometric ratio when successive improvements occur, as follows.

Let $Fix(h)$ denote the number of new variables that are assigned fixed values and added to the backbone at level h. We begin with a chosen value $Fix1$ for $Fix(1)$, referring to the first level at which a backbone is determined, and then generate values for higher levels by making use of an "attenuation fraction" g as follows.

$Fix(1) = Fix1$
$Fix(h) = Fix(h-1) \cdot g$ for $h > 1$

We select the value $Fix1 = 0.25n$ and the fraction $g = 0.4$ as indicated in Table 1.

Within Algorithm 1, we implicitly start with $h = 0$ in the Initialization, and then increment h by 1 at each application of the Rule for Fixing Variables. To be precise, the value $Fix(h)$ is embedded within Algorithm 1 as follows:

Beginning of Rule for Fixing Variables
$h := h + 1$
$na = Fix(h)$

It should be noted that the number of variables selected to be fixed as backbone variables is critical to our BGTS algorithm. If this number is too large, the number of variables potentially fixed at incorrect values can be large enough to prevent the current solution trial from finding a good (improved) solution within a reasonable amount of time. If this number is too small, the convergence of the search may be unacceptably slow.

2.5 Freeing Procedure

Our experiments indicate that in most cases, the fixed backbone variables match well with the putative optimal solution. However, it is still possible that some of these variables are wrongly fixed, resulting in a loss of effectiveness of the algorithm. In order to tackle this problem, it is imperative to free the wrongly-fixed backbone variables so that the search procedure can be put on the right track.

Similar to the fixing procedure, our freeing procedure also consists of three steps:

1. *Scoring*: give each current backbone variable a score;
2. *Selecting*: choose a certain number of current backbone variables;
3. *Freeing*: free the selected variables so that they are allowed to receive random assignments upon launching the next round of TS.

2.5.1 Variable Scoring

The freeing procedure uses the same scoring methods as the fixing procedure. The scored variables are then sorted according to their scores in a non-increasing order. The obvious difference is that in the freeing phase we only consider current backbone variables.

2.5.2 Variable Selection and freeing

Contrary to the fixing phase, the number of the backbone variables released from their assignments at each freeing phase is not adjusted, due to the fact that at each trial only a small number of backbone variables, generally less than five, are wrongly fixed and need to be freed. Specifically, we set the number nd of backbone variables to be freed to equal the value r, as shown in Table 1. To be precise, the value nd is embedded within Algorithm 1 as follows. Then, these selected backbone variables are free to receive new values when initiating the next round of TS.

Beginning of Rule for Freeing Variables
    $nd = r$
    If $nd > nb$ then
        $nd = nb$
    Endif

Table 1 Settings of important parameters

Parameters	Section	Description	Value
tl	2.2	tabu tenure constant	0.007n
α	2.2	tabu search improvement cutoff	100000
p	2.3.2	size of the reference solution population	20
$Fix1$	2.4.2	number of backbone variables at the first fixing	0.25n
g	2.4.2	backbone geometric coefficient	0.4
r	2.5.2	number of freeing backbone variables	60

3 Computational Results

3.1 Problem Instances and Experimental Protocol

To assess the performance of our BGTS algorithm, we use a set of 21 large and difficult random instances with 3000 to 7000 variables. These instances are initially introduced in (Palubeckis (2004)) and recently used in (Glover et al (2010); Lü et al (2010); Palubeckis (2006)) and several other studies. As indicated in (Palubeckis (2004)), these instances are known to be much more challenging than those from ORLIB.

Our BGTS algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.83GHz CPU and 2GB RAM. Table 1 gives the descriptions and settings of the parameters used in the BGTS algorithm for the experiments. Given the stochastic nature of the algorithm, each problem instance is independently solved 20 times.

In accordance with (Glover et al (2010); Lü et al (2010); Palubeckis (2004, 2006)), we use a time limit as our stopping condition. Specifically, the time limit is set to be 5, 10, 30, 60, 60 minutes for instances with 3000, 4000, 5000, 6000 and 7000 variables.

3.2 Computational Results

We present in Table 2 the computational results of the two versions of BGTS (which we denote by BGTS-SS and BGTS-SP). The first column identifies the problem instance. The size of the instance is equal to the number appearing in its name. Columns 2 and 3 respectively give the density (dens) and the previous best objective values (f^*) reported in Glover et al (2010). The remaining columns give the results of one of the two versions according to four criteria: (1) the best solution gap, g_{best}, to the previous best known objective values (i.e., $g_{best} = f^* - f_{best}$ where f_{best} denotes the best objective value obtained by our algorithm), (2) the average solution gap, g_{avr}, to the previous best objective values (i.e., $g_{avr} = f^* - f_{avr}$ where f_{avr} represents the average objective value), (3) the success rate, suc, for reaching the best known result f^* and (4) the average CPU time, t_{avr} (in seconds), for reaching the best result f^*. Furthermore, the last row "Average" indicates the summary of the algorithm's average performance.

Table 2 Computational results on 21 large instances using the SS and SP Algorithms

Instance	dens	f^*	BGTS-SS				BGTS-SP			
			g_{best}	g_{avr}	suc	t_{avr}	g_{best}	g_{avr}	suc	t_{avr}
p3000.1	0.5	3931583	0	6	19	65	0	141	17	104
p3000.2	0.8	5193073	0	0	20	51	0	0	20	53
p3000.3	0.8	5111533	0	36	19	87	0	124	17	90
p3000.4	1.0	5761822	0	39	18	96	0	0	20	98
p3000.5	1.0	5675625	0	138	15	169	0	143	13	149
p4000.1	0.5	6181830	0	0	20	62	0	0	20	75
p4000.2	0.8	7801355	0	625	11	190	0	553	13	201
p4000.3	0.8	7741685	0	11	18	133	0	7	18	248
p4000.4	1.0	8711822	0	0	20	170	0	3	19	111
p4000.5	1.0	8908979	0	907	9	298	0	1125	8	336
p5000.1	0.5	8559680	0	600	4	556	0	663	3	729
p5000.2	0.8	10836019	0	542	7	1129	0	788	5	366
p5000.3	0.8	10489137	0	274	6	874	0	1049	2	786
p5000.4	1.0	12252318	0	912	1	379	0	1735	1	648
p5000.5	1.0	12731803	0	99	14	629	0	91	14	427
p6000.1	0.5	11384976	0	796	3	597	0	529	4	788
p6000.2	0.8	14333855	0	630	4	428	0	1735	3	944
p6000.3	1.0	16132915	0	1432	4	601	0	1954	4	1035
p7000.1	0.5	14478676	0	1606	2	1836	0	1835	2	2704
p7000.2	0.8	18249948	0	2387	2	1569	0	2192	1	1031
p7000.3	1.0	20446407	0	2316	5	703	0	1568	5	1197
Average			0	636	10.5	507	0	773	10.0	572

Table 2 shows that the two versions of our BGTS algorithm can easily reach the previous best known objective values within the given time limit for all the considered instances. Additionally BGTS-SS version performs slightly better than BGTS-SP version relative to the other three criteria, i.e., in terms of the average solution gaps, the success rate and the average CPU time for reaching the best known solutions. In sum, both versions of our BGTS algorithm are efficient in finding the best known objective values for these 21 large difficult instances.

3.2.1 Comparison between BGTS and its underlying TS

We now assess the effect of backbone strategies on the performance of TS by comparing our BGTS algorithm with its underlying TS procedure on the set of 21 instances. For this purpose, we run TS procedure described in Section 2.2 under the same time limit as our BGTS algorithm. The results are shown in Table 3. From Tables 2 and 3, one observes that the two versions of the BGTS algorithm do boost the performance of the basic TS in terms of the criteria (1)-(4) for almost all the instances.

Specifically, when it comes to the best solutions obtained, unlike BGTS-SS and BGTS-SP, the basic TS cannot find the best known values for 4 instances (5000.4, 6000.2, 7000.1 and 7000.2) and the best solution gap (g_{best}) is 118, in comparison with BGTS-SS and BGTS-SP's gap of 0. Furthermore, the average CPU time for BGTS-SS and BGTS-SP to find the best solution is respectively 507 and 572 seconds which is 50% and 44% less than that of the basic TS.

Table 3 Computational results on 21 large instances using the basic TS Algorithm

Instance	*dens*	f^*	Basic TS Algorithm				
			f_{best}	g_{best}	g_{avr}	suc	t_{avr}
p3000.1	0.5	3931583	3931583	0	207	12	50
p3000.2	0.8	5193073	5193073	0	306	12	29
p3000.3	0.8	5111533	5111533	0	679	12	67
p3000.4	1.0	5761822	5761822	0	394	18	44
p3000.5	1.0	5675625	5675625	0	675	5	61
p4000.1	0.5	6181830	6181830	0	13	16	76
p4000.2	0.8	7801355	7801355	0	1766	5	108
p4000.3	0.8	7741685	7741685	0	526	9	204
p4000.4	1.0	8711822	8711822	0	175	14	231
p4000.5	1.0	8908979	8908979	0	1148	11	323
p5000.1	0.5	8559680	8559680	0	925	1	1650
p5000.2	0.8	10836019	10836019	0	1628	1	23
p5000.3	0.8	10489137	10489137	0	2799	2	869
p5000.4	1.0	12252318	12251403	915	2202	0	1800
p5000.5	1.0	12731803	12731803	0	1011	3	531
p6000.1	0.5	11384976	11384976	0	1097	4	1244
p6000.2	0.8	14333855	14333257	598	3180	0	3600
p6000.3	1.0	16132915	16132915	0	1642	6	2279
p7000.1	0.5	14478676	14477845	831	2400	0	3600
p7000.2	0.8	18249948	18249799	149	2875	0	3600
p7000.3	1.0	20446407	20446407	0	4426	2	1134
Average				118	1432	6.34	1025

In addition, BGTS-SS and BGTS-SP's success rate (10.5 and 10 times over 20 runs) to reach the best-known values is about 67% higher than that of the basic TS (6.34/20). Finally, BGTS-SS and BGTS-SP obtain better average objective values (636 and 773 against 1432).

3.2.2 Comparison with an Iterated TS

We also provide a comparison between BGTS and an Iterated TS procedure which reinforces the previous TS procedure with a perturbation-based diversification strategy. Specifically, after each TS run, ITS partially "dismantles"the best local optimum solution obtained by TS, i.e., 1/3 of the variables of the best solution obtained by TS are randomly flipped while keeping other remaining variables unchanged. Tables 2 and 4 show that the two versions of the BGTS algorithm perform better than ITS in terms of three of the four criteria (best solution gaps, average CPU time to find the best solutions and the success rate in reaching best known objective values). This demonstrates the advantage of using backbone information to guide the search over the random perturbation strategy.

3.2.3 Performance comparison under longer time limit

We now investigate the performance of BGTS when a longer time limit is allowed. In this experiment, we only consider the 11 instances with 5000 to

Table 4 Computational results on the 21 large instances using ITS Algorithm

Instance	*dens*	f^*	ITS Algorithm				
			f_{best}	g_{best}	g_{avr}	*suc*	t_{avr}
p3000.1	0.5	3931583	3931583	0	268	17	76
p3000.2	0.8	5193073	5193073	0	49	18	45
p3000.3	0.8	5111533	5111533	0	177	15	75
p3000.4	1.0	5761822	5761822	0	0	20	63
p3000.5	1.0	5675625	5675625	0	675	12	128
p4000.1	0.5	6181830	6181830	0	228	20	61
p4000.2	0.8	7801355	7801355	0	1014	10	166
p4000.3	0.8	7741685	7741685	0	133	14	149
p4000.4	1.0	8711822	8711822	0	123	16	225
p4000.5	1.0	8908979	8908979	0	63	17	247
p5000.1	0.5	8559680	8559355	325	853	0	1800
p5000.2	0.8	10836019	10836019	0	1084	1	370
p5000.3	0.8	10489137	10489137	0	1155	7	650
p5000.4	1.0	12252318	12251874	444	1295	0	1800
p5000.5	1.0	12731803	12731803	0	581	8	472
p6000.1	0.5	11384976	11384976	0	415	8	1961
p6000.2	0.8	14333855	14333855	0	292	6	978
p6000.3	1.0	16132915	16132915	0	793	8	1686
p7000.1	0.5	14478676	14478638	38	954	0	3600
p7000.2	0.8	18249948	18249844	104	1155	0	2304
p7000.3	1.0	20446407	20446407	0	1004	10	2283
Average				43	586	9.86	911

7000 variables. Each instance is independently solved 20 times with a time limit three times that allowed in previous experiments, i.e., 90, 180 and 180 minutes respectively for instances with 5000, 6000 and 7000 variables. For the two BGTS variants, the best solution gaps to the best known objective values (g_{best}), the success rate to reach the best known objective values (*suc*) and the average solution gaps to the best known objective values (g_{avr}) are shown in Table 5.

Table 5 shows that both BGTS versions can find all the best known objective values within the given time limit. Moreover, the success rate for finding the best known objective values is further improved compared to the outcomes in Table 2, equalling 6.4 and 5.5 out of 20 runs, respectively for SS and SP, compared to 4.7 and 4.0 out of 20 runs for the shorter time limit. In addition, the high performance of the BGTS algorithm is more pronounced in terms of the average solution gaps. In fact, the two versions of our BGTS algorithm SS and SP have an average solution gap of 468 and 608 against 1054 and 1285 of the previous experiments in Table 2 for these 11 largest instances.

4 Discussion and Analysis

We now turn our attention to analyzing several important features of the proposed BGTS algorithm, including the number of the backbone variables at each fixing or freeing phase and the average gap to the best known objective values.

Table 5 Computational results with longer time limit using SS and SP Algorithm

Instance	f^*	BGTS-SS			BGTS-SP		
		g_{best}	suc	g_{avr}	g_{best}	suc	g_{avr}
p5000.1	8559680	0	4/20	446	0	1/20	486
p5000.2	10836019	0	12/20	204	0	7/20	384
p5000.3	10489137	0	6/20	206	0	7/20	273
p5000.4	12252318	0	4/20	581	0	4/20	530
p5000.5	12731803	0	19/20	7	0	18/20	20
p6000.1	11384976	0	9/20	152	0	6/20	174
p6000.2	14333855	0	2/20	368	0	3/20	191
p6000.3	16132915	0	3/20	1211	0	4/20	1313
p7000.1	14478676	0	2/20	659	0	2/20	984
p7000.2	18249948	0	1/20	1165	0	1/20	1023
p7000.3	20446407	0	8/20	153	0	8/20	1311
Average		0	6.4/20	468	0	5.5/20	608

4.1 Number of Fixing Backbone Variables

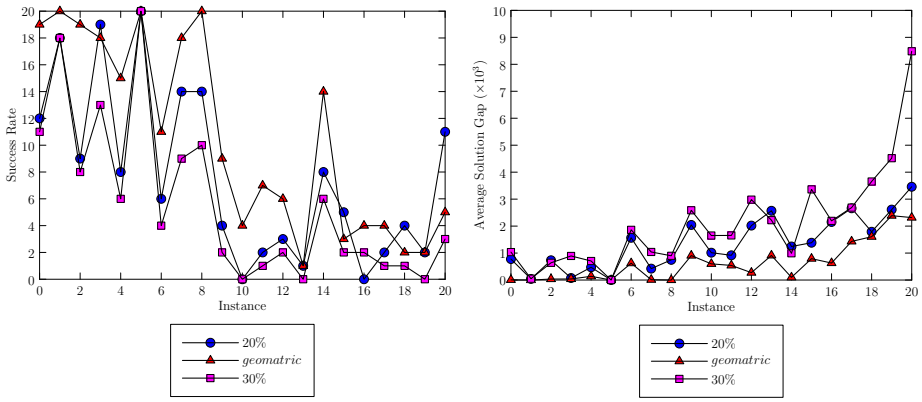

Fig. 1 Succ Rate(L) and Average Gap(R) with different backbone fixing strategies

As indicated in Section 2.4.2, we set the number of backbone variables by employing a geometric ratio reduction strategy. In order to make sure that this strategy is meaningful, we conduct an experiment to compare this strategy with two other linear percentage strategies. Specifically, at each trial the number of backbone variables to be fixed is set to be a constant linear percentage of the total number of all non-backbone variables. In our experiment, we use two linear percentage values: 20% and 30%. We compare the success rate and the average gap to reach the best known values for all the 21 tested instances using the geometric strategy and the two linear percentage strategies, as shown in Figure 1(L). Note that this experiment is carried out using the SS algorithm on all the 21 instances.

One notices that the geometric ratio strategy can obtain the best known objective values (i.e. with a success rate larger than 0) for all the 21 instances,

while the two linear percentage strategies (denoted by 20% and 30%) fail for two and three instances, respectively. In addition, for most of these 21 instances, the success rate of the geometric ratio strategy is greater than that of the two linear percentage strategies, which implies that the geometric ratio strategy is more stable in reaching the best known objective values.

When it comes to the average gaps to the best known objective values, Figure 1(R) shows that the geometric ratio strategy outperforms the two linear percentage strategies for almost all the 21 instances.

4.2 Evolving Average Solution Gap

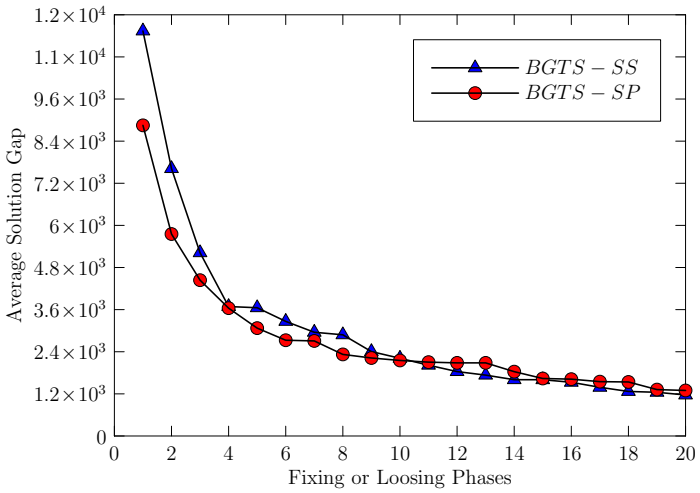

Fig. 2 Evolving solution gaps to the best known objective values

In this section, we carry out another experiment to verify whether the best solution of our BGTS algorithm can be continuously improved with the progress of the search. For this purpose, Figure 2 shows how the best solution gap evolves with the iterations of fixing or freeing phases on the instance p5000.4. This experiment is conducted with both the SS and SP versions of BGTS. Specifically, we calculate the average of the best solution gap to the best known objective value over 10 independent runs. (only the first 20 phases of fixing or freeing variables are indicated).

Figure 2 discloses that the solution gaps for both SS and SP decrease dramatically during the first 6 phases. Moreover, in the remaining phases the objective value is further improved, which indicates that our BGTS algorithm can consistently improve the solution quality when the search progresses, showing the strong search potential of the backbone-guided tabu search algorithm.

5 Conclusions

The backbone-guided tabu search algorithm for the UBQP problem alternates between a basic TS procedure and a variable fixing/freeing phase guided by backbone information based on identifying strongly determined variables. While the TS phase ensures the exploitation of a search space, the variable fixing (freeing) phase dynamically enlarges (reduces) the backbone of assigned values that launches the TS exploration.

To choose the variables to be fixed or freed, the proposed method applies a dedicated scoring mechanism to variables of reference solutions. A geometric ratio strategy is incorporated to determine the appropriate number of backbone variables to be fixed. In total, two versions of the BGTS algorithm are investigated that embody different possibilities for building reference solution.

Using a set of 21 well-known difficult instances with 3000 to 7000 variables, we show that the BGTS algorithm obtains highly competitive outcomes in comparison with the previous best known results from the literature. A direct comparison between BGTS and the underlying TS procedure confirms that incorporating backbone information boosts the performance of the basic TS algorithm.

This research establishes the merit of our backbone guided search for solving the UBQP problem. Future studies can enhance the basic strategy in two key ways by drawing further on the ideas underlying the original proposal for exploiting strongly determined variables: (i) including consideration of consistent variables by reference to the frequency that variables receive assigned values in high quality solutions, and (ii) compelling variables to remain fixed at their selected values for some period during the improving phase (mediated here by tabu search) instead of simply using these values to launch the improving phase.

Acknowledgment

The work is partially supported by a “Chaire d’excellence” from “Pays de la Loire” Region (France) and regional RaDaPop and LigeRO projects (2009-2012).

References

Alidaee B, Kochenberger GA, Ahmadian A (1994) 0-1 quadratic programming approach for the optimal solution of two scheduling problems. International Journal of Systems Science 25:401–408

Alidaee B, Kochenberger GA, Lewis K, Lewis M, Wang H (2008) A new approach for modeling and solving set packing problems. European Journal of Operational Research 86(2):504–512

Amini M, Alidaee B, Kochenberger GA (1999) A scatter search approach to unconstrained quadratic binary programs, McGraw-Hill, New York, pp 317–330. New Methods in Optimization

Borgulya I (2005) An evolutionary algorithm for the binary quadratic problems. Advances in Soft Computing 2:3–16

Boros E, Hammer PL, Tavares G (2007) Local search heuristics for quadratic unconstrained binary optimization (qubo). Journal of Heuristics 13:99–132

Boros E, Hammer PL, Sun R, Tavares G (2008) A max-flow approach to improved lower bounds for quadratic 0-1 minimization. Discrete Optimization 5(2):501–529

Chardaire P, Sutter A (1994) A decomposition method for quadratic zero-one programming. Management Science 41(4):704–712

Garey MR, Johnson DS (1979) Computers and intractability: A guide to the theory of NP-completeness. Freeman, New York

Glover F (1977) Heuristics for integer programming using surrogate constraints. Decision Sciences 8(1):156–166

Glover F (2005) Adaptive memory projection methods for integer programming. In: Rego C, Alidaee B (eds) Metaheuristic Optimization Via Memory and Evolution, Kluwer Academic Publishers, pp 425–440

Glover F, Hao JK (2010) Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. International Journal of Metaheuristics 1(1):3–10

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

Glover F, Kochenberger GA, Alidaee B (1998) Adaptive memory tabu search for binary quadratic programs. Management Science 44:336–345

Glover F, Alidaee B, Rego C, Kochenberger GA (2002) One-pass heuristics for large-scale unconstrained binary quadratic problems. European Journal of Operational Research 137:272–287

Glover F, Lü Z, Hao JK (2010) Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR-A Quarterly Journal of Operations Research doi: 10.1007/s10732-010-9128-0

Harary F (1953) On the notion of balance of a signed graph. Michigan Mathematical Journal 2:143–146

Horst R, Pardalos PM, Thoai NV (2000) Introduction to Global Optimization. Kluwer Academic Publishers, Boston

Katayama K, Narihisa H (2001) Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research 134:103–119

Kilby P, Slaney JK, Thiebaux S, T (2005) Backbones and backdoors in satisfiablity. In: Proceedings of AAAI-2005, pp 1368–1373

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

Kochenberger GA, Glover F, Alidaee B, Rego C (2005) An unconstrained quadratic binary programming approach to the vertex coloring problem. Annals of Operations Research 139:229–241

Krarup J, Pruzan A (1978) Computer aided layout design. Mathematical Programming Study 9:75–94

Lewis M, Kochenberger GA, Alidaee B (2008) A new modeling and solution approach for the set-partitioning problem. Computers and Operations Research 35(3):807–813

Lewis M, Alidaee B, Glover F, Kochenberger GA (2009) A note on xqx as a modelling and solution framework for the linear ordering problem. International Journal of Operational Research 5(2):152–162

Lü Z, Glover F, Hao JK (2010) A hybrid metaheuristic approach to solving the ubqp problem. European Journal of Operational Research doi: 10.1016/j.ejor.2010.06.039

McBride RD, Yormark JS (1980) An implicit enumeration algorithm for quadratic integer programming. Management Science 26:282–296

Merz P, Katayama K (2004) Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78:99–118

Monasson R, Zecchina R, Kirkpatrick S, Selman B, Troyansky L (1998) Determining computational complexity for characteristic 'phase transitions'. Nature 400:133–137

Palubeckis G (2004) Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research 131:259–282

Palubeckis G (2006) Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2):279–296

Wilbaut C, Salhi S, Hanafi S (2009) An iterative variable-based fixation heuristic for 0-1 multidimensional knapsack problem. European Journal of Operational Research 199:339–348

Zhang W (2004) Configuration landscape analysis and backbone guided local search. part 1: Satisfiability and maximum satisfiability. Artificial Intelligence 158:1–26