7 TABU SEARCH WITH CRITICAL EVENT MEMORY: AN ENHANCED APPLICATION FOR BINARY QUADRATIC PROGRAMS

Fred Glover[1], Gary Kochenberger[2],
Bahram Alidaee[3] and Mohammed Amini[4]

[1]Graduate School of Business, University of Colorado
Campus Box 419, Boulder, CO 80309, USA.
fred.glover@colorado.edu

[2]College of Business, University of Colorado, Denver, CO 80217, USA.
gkochenberge@castle.cudenver.edu

[3]College of Business, University of Mississippi, University, MS 38677, USA.
alidaee@bus.olemiss.edu

[4]College of Business, University of Memphis, Memphis, TN 38152, USA.
mamini@msuvx2.memphis.edu

Abstract: A recently developed tabu search approach for solving binary quadratic programs has been shown to be highly effective in finding optimal solutions to known difficult problems. Organized chiefly to employ strategic oscillation, based on critical event memory, the method has not only quickly found optimal solutions in those cases where they are known, but has also found high quality (perhaps optimal) solutions to new test problems that are larger than previously reported in the literature.

This paper extends our previous work by introducing a simple but effective scheme for accelerating the evaluation of candidates' moves (for transforming one solution into another), and for updating associated problem information. Accompanying this we also introduce methods for generating advanced starting solutions and additional trial solutions at critical events. Our preliminary com-

putational experiments on a Pentium 200 PC platform show that the method finds optimal solutions (where they are known) in two to six seconds of computation time. We also include tests for problems containing 1000 variables, which are substantially larger than any that appear in the literature prior to this work. Our enhanced approach obtains high quality solutions to these problems within one to four minutes of computation time.

7.1 INTRODUCTION

Recently, a tabu search (TS) method incorporating strategic oscillation has proved highly successful for solving 0-1 Quadratic Programming Problems [6]. A key feature of this method is the use of a *critical event memory* to guide the oscillation process. We examine enhancements of this approach designed to improve its efficiency and to obtain higher quality solutions. As a foundation for this, we introduce a simple but effective procedure for accelerating the evaluation of moves made by the method, and for updating associated problem information. In addition, we introduce methods for generating advanced starting solutions and for creating additional trial solutions at critical events. These developments make it possible to find best-known solutions to benchmark problems more effectively than before. We also include tests on problems that are substantially larger than those previously examined in the literature. Our study of these problems is motivated by the wide range of discrete optimization problems that can be equivalently formulated as a 0-1 Quadratic Program (see, e.g., [1, 2, 4, 9]).

We organize the remainder of the paper as follows. Section 7.2 introduces basic notation and sketches the specialized tabu search approach on which our study is based. Section 7.3 provides the ideas and algorithms that underlie our proposed enhancements. Section 7.4 elaborates our approach for generating advanced starting solutions and trial solutions at critical events. Then, Section 7.5 presents our computational results. Finally, we present our assessments and conclusions.

7.2 NOTATION AND OVERVIEW

We formulate the binary quadratic programming problem (QP) as follows:

$$QP : \max x_0 = xCx \text{ , where } x \text{ is a zero-one vector.}$$

The tabu search approach we employ to solve QP relies chiefly on the strategic oscillation component of TS. The strategic oscillations alternate between constructive phases (progressively setting variables to 1) and destructive phases (progressively setting variables to 0). We refer to the process of setting variables to 1 during a constructive phase as "adding" variables and the process of setting variables to 0 in a destructive phase as "dropping" variables.

To control this form of oscillation we use a memory structure that is updated at *critical events*. For the binary quadratic programs considered here, we define a critical event as when the next move (either an add or a drop) causes the objective function to decrease, hence to worsen in the maximization context.

The corresponding solution is called a *critical solution.* We utilize a parameter, *span*, to indicate the amplitude of oscillation about a critical event, measured by the number of moves made beyond the critical event. We begin with span equal to 1 and gradually increase it to some limiting value ($MaxSpan$). For each value of *span*, a series of alternating constructive and destructive phases is executed before progressing to the next value. At the limiting point, we begin to gradually decrease *span*, allowing again for a series of alternating constructive and destructive phases. When *span* reaches a value of 1, a complete span cycle is concluded and once again a gradual increase in span is initiated, starting the next span cycle.

Information stored at critical events is used to influence the search process in the following manner. We record (in a circular list) the t most recent critical solutions found. From these solutions, we maintain a short term recency vector ($TabuR$) that is the sum of these t solutions. By summing the associated solutions, this memory is not only a recency memory but a frequency memory, since each element indicates the number of times that the associated variable receives a value of 1 in the last t iterations. In a like manner, we maintain a long term frequency vector ($TabuF$) which is the sum of all critical solutions found to date. These arrays are used to influence the search process to head in new directions in pursuit of new critical solutions. This is accomplished by applying the memory to affect the choice of variables to receive new value assignments at the *turn around* points of oscillation, where the method changes from a constructive phase to a destructive phase, or vice versa.

We emphasize that this variant of tabu search embodies only a restricted subset of its ideas, but a subset that has proved effective in the present context. Other elements of tabu search are elaborated, e.g., in [7].

To illustrate the rationale underlying our current approach, suppose we have just switched from the destructive phase to the constructive phase. Other things being equal, we would like to choose the variable (to add next) that contributes the largest net increase to the objective function value x_o. To facilitate a search that appropriately balances intensification and diversification, however, we also would like to choose variables that have not appeared at a value of 1 in recent critical solutions. Thus, we seek to impose the condition that the first variable added back, x_j, will have $TabuR(j) = 0$. That is, we can conceive the condition $TabuR(j) > 0$ as implying that variable x_j is tabu. More generally, we seek to require that the first p variables added (after turn around) to have $TabuR(j) = 0$. Such a requirement might not be strictly possible. Consequently, we attach a positive penalty weight, $PenR$, to $TabuR(j)$ and create a penalized evaluation of x_j by subtracting the penalty term $PenR \cdot TabuR(j)$ from the normal evaluation of variable x_j. Long term information, via $TabuF(j)$, is included in a similar fashion but with a much smaller penalty coefficient.

By a corresponding rationale, when we switch from the constructive phase to the destructive phase, we seek to influence the choice of drops made after this turn-around by using the tabu vectors to create inducements to drop vari-

ables appearing in recent critical solutions at a level of 1. Thus, we proceed as previously described except that the penalty terms are added (rather that subtracted) to determine the evaluation of a variable.

The parameter $NewAssign$, which identifies the number of tabu-influenced adds or drops made immediately after a turn-around, is managed in a fashion that fosters additional diversity in the search process. We start with $NewAssign = 1$, and after a number of iterations (e.g., $2t$), we set $NewAssign = NewAssign+1$. We continue in this fashion until NewAssign reaches a limit ($MaxNewAssign$) at which point we set $NewAssign$ back to 1 and the process repeats.

Between the constructive and destructive phases is the transfer phase which orchestrates the oscillations, manages key parameters, and does various bookkeeping chores. We refer to an execution of the Transfer Phase as an iteration of our heuristic. How the various parameters are managed can make a difference in the performance of the heuristic on certain problems. Given that our goal in this paper was to test the usefulness of ideas for accelerating the method and for generating additional trial solutions (rather than trying to optimize parameters settings), we utilized the following parameter values for all of the computational testing: $t = 3$, $MaxSpan = 7$, $MaxNewAssign = 4$.

7.3 FOUNDATION FOR GENERATING TRIAL SOLUTIONS

7.3.1 Efficient Move Evaluations

Both in the basic operation of our TS method, and in the procedures subsequently described for generating advanced starting solution and trial solutions, we require variables to be evaluated in terms of the net increase in the objective function associated with flipping their value. To facilitate the efficiency of such processes, we employ the following design. Let $N = \{1, ..., n\}$ denote the index set for components of x. Throughout this paper we use the convention that x' and x'' denote two successive trial solutions, where x'' is obtained from x' by identifying a specific index $k \in N$, for which x''_k differs from x'_k while $x''_i = x'_i$ for all $i \in N$, $i \neq k$. Let E denote an *evaluation matrix* given by

$$E_{i,i} = C_{i,i} \text{ for } i \in N$$
$$E_{i,j} = C_{i,j} + C_{j,i} \text{ for } i, j \in N, i \neq j$$

and define $e_i = E_i x, i \in N$, where E_i denotes the ith row of E. We understand e'_i and e''_i to be the form of e_i when $x = x'$ and $x = x''$, respectively.

The evaluation matrix E and the associated quantity e_i provide a basis for conveniently identifying the change in the objective function when x' is replaced by x''. In order to identify the element x'_k of x' that will be changed to yield x'', we will make reference to the change δ_k in x_o defined by $\delta_k = x''_o - x'_o$ (where x''_o and x'_o depend on the choice of k as indicated). Computing this change directly from the definition of δ_k (i.e., $\delta_k = x''Cx'' - x'C'$) requires $O(n^2)$ effort. The following result relies on memory of E and allows δ_k to be

identified for all $k \in N$ with $O(n)$ effort, including the effort to update for e'_i to e''_i all $i \in N$ once the decision is made to replace x' with x''.

Proposition Let x' and x'' be two n-vectors such that $x'_i = x''_i$ for $i \in N, i \neq k$. Then

$$\delta_k = (e'_k + E_{k,k}x''_k)(x''_k - x'_k) \tag{7.1}$$

and the updated value e''_i for e'_i when x'' replaces x' becomes

$$e''_i = e'_i + E_{i,k}(x''_k - x'_k) \text{ for all } i \in N \tag{7.2}$$

Proof First we determine e''_i. From the relation between x' and x'' we have $E_i x'' = E_i x' - E_{k,k}x'_k + E_{k,k}x''_k$. Since $E_i x' = e'_i$ and $E_i x'' = e''_i$ this establishes the updated e''_i value as specified in (7.2). Next, by direct expansion from the definition of δ_k and from the definitions of e'_i and e''_i we can determine that $\delta_k = x''_k e''_k - x'_k e'_k$. Substituting the value for e''_i established in (7.2), for $i = k$, yields $= x''_k(e'_k + E_{k.k}(x''_k - x'_k)) - x'_k e'_k$. Collecting and rearranging terms then yields the expression specified in (7.1).

Under the situation where one of x' and x'' has the value 0 and the other has the value 1, the expressions (7.1) and (7.2) simplify to provide particularly fast determinations of δ_k and e''_i for $i \in N$.

7.3.2 Trial Solution Generation

To extend the scope of our investigation, we will make use of our approaches for generating trial solutions as a basis for generating starting solutions as well. The utility of good starting points for combinatorial optimization problems has been suggested by a variety of studies. For the binary quadratic problem we are considering here, Pardalos and Rodgers [12] used a simple gradient ascent method to identify a starting point to initialize their branch and bound algorithm. Based on computational testing over a wide variety of problems, they comment that their starting point was in fact optimal for roughly 60% of the problems and consistently within 3% of optimum when not optimal.

The straightforward ascent methods we use to generate trial solutions are structured to operate on any *seed solution* as an input, and thus can be used to generate starting solutions by selecting various simple seeds, such as setting all variables to 1 or all variables to 0. The new solution produced by our algorithms differs from the seed solution whenever the seed solution is not locally optimal. We continue to assume, for any given $k \in N$, that x'' is the solution that complements (flips) the value of x'_k while leaving x'_i for $i \neq k$ unchanged.

Method P (Partial Ascent) Let x' be a chosen 0-1 seed solution, and create a 0-1 trial solution x'' by the following rule. For every $k \in N$: If $\delta_k > 0$, let $x^*_k = x''_k$. Otherwise, let $x^*_k = x'_k$.

We call Method P a "Partial" Ascent Method because it operates by identifying the flips that improve the solution x', and making all such flips at once.

The outcome does not guarantee that the trial solution x^* will improve x', since moves that are improving when considered independently may not be improving in combination.

It is possible to show that algorithm 4.2 given by Pardalos and Rodgers [12] (P&R), which is based on gradient arguments, is a special case of Method P when $x' = (1, 1, 1, 1, ..., 1)$ (although the P&R method does not make reference to a seed solution). This connection provides a basis for interesting comparisons, described later.

Method A (Ascent)

1. Let x' be a chosen 0-1 seed solution.
2. Let J be the subset of N defined by $J = \{j \in N : \delta_j > 0\}$.
3. If J is empty, denote the final x' obtained by x^* and stop. Otherwise, choose an element k from J and redefine x' to be the solution x''. Then return to Step 2.

Method A is a classical ascent method, which we implement by taking advantage of expressions (7.1) and (7.2) of the Proposition. For our present purposes we choose k in Step 3 by the "steepest ascent" rule, to give $k = \arg\max(\delta_j : j \in N)$. Finally, we join Method P and Method A to give the following procedure.

Method PA

1. Apply Method P, starting with a chosen seed solution x'.
2. Redefine $x' = x^*$ where x^* is the solution obtained from Method P. Then apply Method A.

7.4 EXPLOITING STARTING SOLUTIONS AND TRIAL SOLUTIONS

There are several possible ways to take advantage of the procedures P, A and PA for generating trial solutions from various seed solutions. We first apply the algorithms P, A and PA to simple initial seed solutions as a basis for generating starting solutions for our method. We are motivated to do this particularly for the PA approach, because in the special case where the seed is given by $x' = (1, 1, ..., 1)$ the resulting solution can be shown to be the same one created by Pardalos and Rodgers (although their method is not defined in terms of seed solutions).

We also use the ascent methods to generate trial solutions at critical events of the strategic oscillation process, where the critical solutions produced at such points are treated as seed solutions. Such an approach can be used either to replace the seed solutions by the resulting trial solutions, or to use the trial solutions as supplementary solutions which do not alter the search process. In our present preliminary study, to provide a more direct comparison with our previous strategic oscillation approach for QP, we rely only on the use of

trial solutions to supplement the search, without altering its progression. The hypothesis to test is that this will make it possible to identify optimal solutions in fewer iterations than by our previous approach, while also providing the ability to find better solutions than previously found in cases where optimal solutions are unknown and may not have been previously obtained.

7.5 COMPUTATIONAL EXPERIENCE

The computational experience reported in this paper is based on the three sets of test problems described in Table 7.1. All problems were obtained using the generator given by Pardalos and Rodgers [11] and all have characteristics designed to make them challenging. (This generator has also been used in the recent heuristic QP study by Chardaire and Sutter [3].) The first 25 problems (the "a", "b" and "c" problems) are small to moderate sized problems which are also reported on by Pardalos and Rodgers. Problem set 2 contains new test problems of intermediate size while problem set 3 contains new test problems much larger than any reported in the literature to date. Optimal solutions for the problems of sets 2 and 3 have yet to be proven.

The branch and bound (B&B) algorithm of Pardalos and Rodgers successfully and rapidly solved the problems of set 1 except for the last three "c" problems. However, the problems of sets 2 and 3 ("d", "e", "f", and "g") have combinations of complexity and dimension that put them beyond the capability of the B&B approach.

The 1000 variable "g" problems are new test problems that are significantly larger than those previously considered in the literature. In fact, the problems of sets 2 and 3 all extend the test bed of problems examined in previous studies. Sets 1 and 2 were used to test the quality of the starting point methods proposed. Then, all three sets were used to test the performance of our TS heuristic enhanced by advanced starts and/or additional trial solutions at critical points. All runs on all problems are carried out with the same set of parameters in our TS heuristic, with no attempt at fine tuning. In every case considered, the problems were run until reaching a limit on the number of complete span cycles. The cycle limit for problems sets 1, 2 and 3 were 20, 40 and 50 cycles, respectively.

Note, that the solutions given for problems "a", "b", and "c" are known to be optimal [12]. Best known solutions to other problems appear later in the paper. All problems were developed using the Pardalos/Rodgers generator [11] with element ranges as given in Table 7.2. The alternative start and trial solution combinations tested are listed in Table 7.3. The seed solutions, the initial x' vectors, used to generate starting solutions in this study are $seed_1$: $x' = (1, 1, 1, 1, ..)$, $seed_2$: $x' = (0, 0, 0, 0, ..)$, $seed_3$: $x' = (1, 0, 1, 0, ..)$, $seed_4$: $x' = (0, 1, 0, 1, ..)$.

Set 1				Set 2			Set 3		
Problem	Size	Den	Solution	Problem	Size	Den	Problem	Size	Den
1a	50	0.1	3414	1d	100	0.1	1g	1000	0.1
2a	60	0.1	6063	2d	100	0.2	2g	1000	0.2
3a	70	0.1	6037	3d	100	0.3	3g	1000	0.3
4a	80	0.1	8589	4d	100	0.4	4g	1000	0.4
5a	50	0.2	5737	5d	100	0.5	5g	1000	0.5
6a	30	0.4	3980	6d	100	0.6	6g	1000	0.6
7a	30	0.5	4541	7d	100	0.7	7g	1000	0.7
8a	100	0.0625	11109	8d	100	0.8	8g	1000	0.8
1b	20	1	133	9d	100	0.9	9g	1000	0.9
2b	30	1	121	10d	100	1	10g	1000	1
3b	40	1	118	1e	200	0.1	11g	1000	0.1
4b	50	1	129	2e	200	0.2	12g	1000	0.2
5b	60	1	150	3e	200	0.3	13g	1000	0.3
6b	70	1	146	4e	200	0.4	14g	1000	0.4
7b	80	1	160	5e	200	0.5	15g	1000	0.5
8b	90	1	145	1f	500	0.1			
9b	100	1	137	2f	500	0.25			
10b	125	1	154	3f	500	0.5			
1c	40	0.8	5058	4f	500	0.75			
2c	50	0.6	6213	5f	500	1			
3c	60	0.4	6665						
4c	70	0.3	7398						
5c	80	0.2	7362						
6c	90	0.1	5824						
7c	100	0.1	7225						

Table 7.1 Problem Sets

Problem	main diagonal	off diagonal
a	$+/-100$	$+/-100$
b	$-63 to 0$	$0 to 100$
c	$+/-100$	$+/-50$
d	$+/-75$	$+/-50$
e	$+/-100$	$+/-50$
f	$+/-75$	$+/-50$
lg-10g	$+/-75$	$+/-50$
11g-15g	$+/-100$	$+/-50$

Table 7.2 Element ranges for the Pardalos/Rodgers generator

Code	Start Method(s)	Trial Solution Method(s)
TS-0	default (zero)	none
TS-1	P, A, PA	none
TS-2	default (zero)	P, A
TS-2	P, A, PA	P

Table 7.3 Alternatives tested, Conditions

7.5.1 *Advanced Starting Points*

The three starting point methods were tested with the four initial seed solutions specified above. The results are presented in Tables 7.4 and 7.5.

As expected, Method P is dominated by Methods A and PA in terms of the quality of starting solutions. The results from Method P are listed in Table 7.4 to give the reader some indication of the method's performance, but they are omitted from Table 7.5 since they are inferior outcomes. Out of the 180 cases reported in Tables 7.4 and 7.5 (25 problems in Table 7.4, 20 problems in Table 7.5, and 4 seed solutions for each), Method PA gives the best starting solution 89 times while Method A gives the best solution 65 times. The two methods tie 26 times. Method PA produces better starting solutions more often than Method A for three of the four seed solutions, but performs slightly worse than Method A for the remaining seed solution (seed solution 2).

Of the 200 starting points generated by Methods A and PA (100 points generated by each) for the relatively easy problems that are considered in Table 7.4, 70 are in fact optimal. Of these, Method PA gives an optimal solution 41 times and Method A 29 times. Conditioning for initial seed solutions, out of the 25 possibilities (in Table 7.4) for each seed, either or both (A and PA) is optimal 19 times for $seed_1$, 14 times for $seed_2$, 24 times for

Table 7.5, which involves a harder set of problems, paints a different picture of starting solution quality. Of the 160 cases presented there, the best starting point generated is equal to the best known solution only three times. While the easier problems of set 1 give rise to optimal starting solutions much more readily than the problems of set 2, even there the percentage of optimal starting solutions is nowhere near the 60% figure referred to by Pardalos and Rodgers [12]. As previously noted, Method PA initialized with $seed_1$ is precisely the starting solution used by Pardalos and Rodgers [12]. Out of the corresponding 25 problems of Table 7.4, the starting points produced are optimal only 11 times. ($seed_3$ gives substantially better results for these easier problems.) And, for the corresponding cases in Table 7.5 for $seed_1$, none of the 25 starting points produced are best known solutions.

7.5.2 *Performance of Advanced Starts and Trial Solutions*

While the quality of various starting points is of interest, our main concern is to gain insight into the extent to which the performance of our TS heuristic may be improved by employing the ideas of Section 7.4, including the use of advanced starts and additional trial solutions. To this end, we solved the problems of all three problem sets with each of the alternatives given in Table 7.3. We refer to our previous TS approach as TS-O, which we have streamlined to take advantage of the efficient updating provided by the Proposition. TS-1 augments TS-0 by including an advanced start, which is always the better of the two starts produced by Method A and PA, using seed 3. TS-2, in contrast, does not employ an advanced start, but rather generates additional trial solutions at critical points using both Methods A and P. Alternative TS-3 is included

	Method			x_o' : $seed_2$ Method			x_o' : $seed_3$ Method			x_o' : $seed_4$ Method		
Prob	P	A	PA	P	A	PA	P	A	PA	P	A	PA
1a	2353	3388	3414*	1599	3404	3414*	2979	3414*	3414*	1360	3355	3414*
2a	4790	6063*	6063*	2246	5967	5989	2943	6063*	6063*	3379	5916	5989
3a	4403	6035	6035	1044	6026	6037*	4025	5949	6033*	3081	5960	6037*
4a	8067	8461	8589	2386	7066	8598*	3178	8598*	8598*	4986	8495	8535
5a	4488	5666	5353	2649	5704	5476	2249	5476	5712	2436	5300	5230
6a	3436	3980*	3980*	1032	3404	3980*	2142	3980*	3846	2304	3980*	3980*
7a	4404	4541*	4541*	1125	4541*	4263	2163	4541*	4541*	3729	4263	4541
8a	8800	11109*	11109*	4245	11031	10965	8075	11109*	11109*	7087	11101	10869
1b	0	98	133*	< 0	133*	98	0	85	133*	0	98	133*
2b	0	121*	91	< 0	91	121*	0	113	91	0	115	91
3b	0	56	102	< 0	102	56	0	88	102	0	118*	102
4b	0	86	101	< 0	101	86	0	84	101	0	118	101
5b	0	70	150*	< 0	150*	70	0	150*	150*	0	54	150*
6b	0	87	113	< 0	113	87	0	105	113	0	106	113
7b	0	90	160*	< 0	160*	160*	0	160*	160*	0	86	160*
8b	0	87	117	< 0	117	87	0	145*	117	0	63	117
9b	0	96	127	< 0	127	99	0	99	127	0	88	127
10b	0	69	121	< 0	121	100	0	128	121	0	111	121
1c	4319	5058*	5058*	430	5058*	5858*	2457	5058*	5058*	2962	5058*	5058*
2c	4943	6213*	6213*	2274	6151	6151	3672	6119	6151	4112	6102	6213*
3c	5941	6665*	6665*	2052	6649	6665*	3922	6665*	6582	3281	6649	6649
4c	5951	7398*	7277	3187	7200	7398*	5316	7398*	7398*	5458	7366	6940
5c	6818	7336	7326	3279	7301	7336	4497	7362*	7326	4320	7272	7272
6c	4981	5762	5707	3699	5718	5745	3926	5694	5762	4040	5697	5763
7c	5916	7124	7211	4431	7015	7173	4418	7147	7087	4493	7225*	7160
Opt	0	8	11	0	5	9	0	12	12	0	4	9

Table 7.4 Starting Point Results: Problem Set 1. All values marked with an asterisk are optimum x_o values. All computation times are less than 1 second.

	x_o' : $seed_1$ Method		x_o' : $seed_2$ Method		x_o' : $seed_3$ Method		x_o' : $seed_4$ Method	
Prob	P	PA	P	PA	P	PA	P	PA
1d	6310	6272	6275	6054	6248	6232	6272	6333*
1d	6534	6381	6217	6050	5722	6267	6040	6673
3d	9205	9108	8647	9056	9193	8969	9085	9140
4d	10660	10548	10629	10714	10659	10461	9893	9974
5d	11421	11491	11604	11477	11410	11565	11290	11474
6d	14024	13981	13946	13762	14207*	14207*	13581	14206
7d	13686	13950	13806	13800	13349	13762	14384	14311
8d	16238	16143	16136	16093	15942	16216	16149	16128
9d	15089	15584	15588	15588	15417	15458	15557	15333
10d	19039	18901	18962	18198	18497	18538	19018	18961
1e	16379	16403	16289	16419	15895	16013	16284	16362
2e	23179	23312	23133	23031	22405	23245	22982	23203
3e	24775	24793	24445	24312	24060	24270	23129	23501
4e	35373	35373	35578	35424	34532	35431	35317	35366
5e	34387	35039	34827	34046	34114	34601	34403	34895
1f	60427	60005	60047	59953	59879	60276	59949	60441
2f	99150	99759	97193	97489	98003	98424	99310	97062
3f	135935	136678	135456	134903	133702	136520	134564	134457
4f	170028	171184	169719	168263	170763	170519	169806	170930
5f	187803	189291	186672	187760	186737	188117	186505	185695
best	0	0	0	0	1	1	0	1

Table 7.5 Starting Point Results: Problem Set 2. All values marked with an asterisk are optimum x_o values. All computation times are less than 1 second.

to give some indication of the usefulness of allowing for both advanced starts and additional trial solutions. It uses only the weakest (although fastest) of the methods to generate trial solutions, Method P.

The results of our runs are shown in Table 7.6. These tables indicate, for each alternative tested, the best objective function value found in the cycle limit allowed and the iteration count at which the best value was found. The bottom of each table also shows average run times for the various problems. For Alternatives TS-2 and TS-3, the tables also indicate (next to the iteration count) the procedure for generating trial solutions that produced the best result. For example, for problem 3a, TS-2 gave the optimal solution at iteration 74 via Method A, while TS-3 gave the optimal solution at iteration 110 via Method P. When no trial method is shown, the associated trial procedure did not produce the best result. The average runtimes are given in Table 7.7.

Results for Problem Set 1 Referring to Table 7.6, we see that all four alternatives produced the optimal solution to each of the 25 problems of Set 1. Comparing the results we see general improvement (in iteration count) by the alternatives to TS-0 in certain cases but degradation of performance in others. These problems were very easy for our TS-0 procedure and, aside from the fact that some of the problems were immediately solved due to optimal starting points, little improvement was anticipated to be possible on the other problems. Solution times are quite modest in all cases.

Compared to the base case results of TS-0, method TS-2 (which generates trial solutions) generally gave somewhat reduced iteration counts. For those problems for which the starting point was not optimal, TS-1 and TS-3 often required more iterations than TS-0. In several cases where the starting point was not optimal, initializing with an advanced starting solution actually impeded the search process. For example, the optimal solution of problem 3a is 6037. The best starting point for this problem yields an objective function value of 6033. Yet, initializing TS-1 at this starting point causes the TS heuristic to run for 603 iterations before finding an optimal solution. In contrast, the TS-0 heuristic found the optimal solution at iteration 127. Table 7.6 displays other examples of this behavior as well.

Results for Problem Set 2 As shown in Table 7.6, an advanced starting solution matched the "best known" solution only once (problem 6d) for Set 2. While the various methods generally produced the same "best" results, there are some differences. Taking TS-0 as the reference, we see that TS-1 (which uses an advanced start) produced the same results 18 times, an inferior result 1 time, and a better result 1 time. TS-2 (which employs trial solutions) gave the same results as TS-0 for all 20 problems, but often found these solutions in significantly fewer iterations. Finally, in terms of solution quality, TS-3 produced the same result as TS-0 19 times and a superior result 1 time.

	TS-0 (Base case)		TS-1		TS-2		TS-4	
Problem	best	iter	best	iter	best	iter	best	iter
				Problem Set 1				
1a	3414	25	3414	0	3414	4P	3414	0
2a	6063	5	6063	0	6063	4A	6063	0
3a	6037	127	6037	603	6037	74A	6037	110P
4a	8598	99	8598	0	8598	25A	8498	0
5a	5737	107	5737	41	5737	1A	5737	8P
6a	3980	12	3980	0	3980	3A	3980	0
7a	4541	2	4541	0	4541	1P	4541	0
8a	11109	30	11109	0	11109	1A	11109	0
1b	133	1	133	0	133	1	133	0
2b	121	60	121	612	121	8A	121	612
3b	118	51	118	51	118	12A	118	51
4b	129	2	129	2	129	1A	129	2
5b	150	1	150	0	150	1	150	0
6b	146	28	146	54	146	27A	146	54
7b	160	1	160	0	160	1	160	0
8b	145	61	145	0	145	61	145	0
9b	137	1686	137	2666	137	682A	137	2665P
10b	154	55	154	30	154	10A	154	30
1c	5058	1	5058	0	5058	1	5058	0
2c	6213	3	6213	13	6213	3	6213	13
3c	6665	36	6665	0	6665	1A	6665	0
4c	7398	77	7398	0	7398	9A	7398	0
5c	7362	289	7362	0	7362	6A	7362	0
6c	5824	293	5824	79	5824	9A	5824	79
7c	7225	27	7225	217	7225	6A	7225	12P
				Problem Set 2				
1d	6333	604	6333	26	633	15A	6333	26
2d	6579	50	6579	320	6579	26A	6579	157P
3d	9261	26	9261	26	9261	25A	9261	26
4d	10727	77	10727	29	10727	37A	10727	29
5d	11626	29	11626	529	11626	4A	11626	529
6d	14207	44	14207	0	14207	43A	14207	0
7d	14476	89	14476	33	14476	89	14476	13
8d	16352	98	16352	26	16352	11A	16352	26
9d	15656	4	15656	30	15656	3A	15656	30
10d	19102	5	19102	30	19102	1A	19102	30
1e	16464	914	16458°	1265	16464	203A	16464	615P
2e	23395	2043	23395	532	23395	109A	23395	202P
3e	25243	63	25244	75	25243	46A	25243	75
4e	35594	82	35594	87	35594	3A	35594	36P
5e	35154	1443	35154	1204	35154	396A	35154	244P
1f	61194	434	61194	347	61194	433A	61194	347
2f	100161	779	100161	3592	100161	42A	100161	3592
3f	138035	126	138035	679	138035	126	138035	679
4f	172618	1811	172771*	1254	172618	1188A	172771*	1254
5f	190507	297	190507	854	190507	297	190507	339P
				Problem Set 3				
1g	131456	2863	131448°	1471	131456	1992A	131456	1198P
2g	172788	4566	172788	5584	172788	2244A	172788	5584
3g	192565	199	192565	201	192565	195A	192565	201
4g	215010	6511	215010	1269	215010	166A	215010	1269
5g	242367	1249	242367	4852	242367	1248A	242367	4852
6g	243393	5663	240569°	1469	243293	5663	240569°	1332P
7g	252664	4264	252466°	1439	252664	4213A	252466°	1438P
8g	264061	3314	263637°	1707	264061	3312A	263637°	1707
9g	261792	3053	262658*	2026	261792	2856A	262658*	2026
10g	273434	3337	273795*	5993	273434	3325A	273795*	5993
11g	131623	1883	131615°	1932	131631*	1379A	131631*	1387P
12g	173615	900	173523°	4039	173615	900	173529°	3690P
13g	192854	218	192864	462	192864	217A	192864	462
14g	215611	6029	215852*	4537	215612*	1090A	215852*	4443P
15g	242832	2573	242832	631	242832	2572P	242832	631

Table 7.6 Starting Point Results: Problem Set 2, Set 2 and Set 3. An iteration count of zero means the starting point was optimal. Each TS variant above found an optimal solution to every problem of Problem Set 1. Here, * denotes solution better than base case, while ° denotes solution worse than base case.

Problems	TS-0	TS-1	TS-2	TS-3
"a"	2	2	4	3
"b"	2	2	5	2
"c"	2	2	6	3
"d"	4	4	15	10
"e"	8	8	43	24
"f"	21	21	320	133
"g"	65	67	1126	424

Table 7.7 Average time for 20 cycles for Problem Set 1, 40 cycles for Problem Set 2 and 50 cycles for Problem Set 3 (in seconds on a Pentium 200 PC)

Results for Problem Set 3 For Problem Set 3 (Table 7.6) we see more variability than for Problem Set 2. Again comparing to TS-O, TS-1 gave the same "best" solution 6 times, an inferior solution 6 times, and a better solution 3 times. In contrast, TS-2 gave the same result 14 times and a better result 1 time. Finally, TS-3 gave the same result 7 times, an inferior result 4 times, and a superior result 4 times. Once again we see general improvement in iteration count (over the base case), but isolated cases of enlarged iteration counts were found.

General remarks on the results for the problem sets A glance at Table 7.6 indicates how frequently Method A produced the best solution. For the 60 problems for which Method A was employed to produce trial solutions, it gave the best solution 48 times. Excluding those cases where the starting point was optimal, Method P was employed on 105 problems and produced the best result 20 times.

Looking at our experience with all 60 problems and the various testing carried out, Method A appears to be the preferred approach for generating additional trial solutions. However, the base case TS-0 method, with no advanced start and no additional trial solutions, gave the best overall performance in terms of solution quality and execution time, even though it generally required more iterations. Alternative TS-2 (which included trial solutions) seldom improved upon the solution quality of TS-0 and yet required substantially greater computation times. Alternatives TS-1 and TS-3, while improving upon the TS-0 solution 5 times, were just as likely to produce inferior solutions.

Since the main source of improved solutions in the preceding tests comes from the use of the advanced trial solution, but the outcomes also include results worse than those for the base case TS-O, we were motivated to test the result of using advanced solutions based on all four of our seed solutions. Since this increases the solution time by a factor of 4, we also investigate the option of simply running TS-0 for four times as many span cycles.

The outcomes appear in Table 7.8. The tests are limited only to the 1000 variable problems. We see that running TS-0 four times longer finds two "new

	BQ1		BQ2		
Problem	best	iter	best	iter	seed#
1g	131456	2863	131456#	5217	4
2g	172788	4566	172788	2234	2
3g	192565	199	192565	201	3
4g	215679*	14067	215679*	6475	4
5g	242367	1249	242368	629	4
6g	243293	5663	243293#	365	4
7g	253181*	12558	253181*	1010	2
8g	264061	3314	264061#	2547	4
9g	261792	3053	262658(+)	2026	3
10g	273719#	18226	274264*(+)	2982	1
11g	131631#	8093	131632	6858	1
12g	173615	900	173542(−)	5908	2
13g	192864	218	192865	296	2
14g	215685#	27606	216012*(+)	6073	4
15g	242832	2473	242833	492	2

Table 7.8 Multiple Starts vs. Base Case. Again, * denotes new best solution (over all runs referenced in this paper), # denotes better than previous best for corresponding method, (+) denotes better than base case result and (−) denotes inferior to base case result. The base case results (BQ1) were run for 200 cycles. The average run times (for 200 cycles) was approximately four minutes for each problem.

best" solutions out of the fifteen problems tested - that is, two solutions better than obtained by any of the previous methods and solution attempts. The method also improves the solution of three additional problems by comparison to the version of TS-0 that was run for fewer iterations. (In fact, as shown in the concluding section, we obtain still better results for some of these problems by a simple variation.)

The BQ2 results show the best objective function value found via initializing BQ2 with the four different seeds. For each seed, the heuristic was run for 50 cycles. The combined time for all 200 cycles was approximately four minutes for each problem.

The method TS-1, which restarts from the four different seeds (and uses the same total number of iterations as the corresponding TS-0 from Table 7.8), eliminates all but one of the cases where TS-1 previously was worse than TS-0, and additionally finds four solutions better than the best previously found by any of the methods. (These match the two new best found by TS-1, and include two others.) Overall, TS-1 obtains three solutions that are better than TS-0 and one solution that is worse. This suggests that there may be value in generating a larger collection of appropriately diverse solutions for re-starting. Alternatively, it may indicate the possibility that the use of longer term frequency memory within TS-0, which serves also to create diversity, should be refined to obtain better outcomes. Frequency memory can also take a role in determining new starting solutions. We stress again that no effort has been made to tune any parameters of the current method, which may offer an opportunity for refinement. The three parameter settings used are the same as those used to solve the smaller and easier problems from our earlier study.

7.6 CONCLUSIONS

Our study has provided several findings. Advanced starting solutions, based on the use of simple seeds (which provide methods more general than those previously used to generate advanced starts), have a widely variable and inconsistent effect on the performance of procedures that use these solutions.

None of the seeds tested produced outcomes that dominated those produced by other seeds. A seed that set all variables equal to 1, which is motivated as "best" by gradient arguments, performed notably worse than a seed that alternately set variables to 0 and 1.

Contrary to reports of a B&B study that starting solutions based on gradient arguments produced optimal solutions in 60% of the cases, we found that such solutions were optimal in less than half of the cases - when these cases were restricted to problems that are easy to solve. For harder problems, these starting solutions (and starting solutions generated by alternative seeds) were almost never optimal.

Trial solutions using the same methods employed for generating starting solutions, but using critical solutions provided by the TS method as seeds, often uncovered previously best known solutions in substantially fewer iterations than required by the TS method that did not make use of these trial solutions. However, these trial solutions did not find solutions superior to the best previously found, and the computation time required to generate these solutions significantly increased the overall solution time of the method.

Allowing the base case method TS-0 to run four times longer (which still consumes only about 4 minutes for each of the large 1000 variable problems), generates improved solutions for 1/3 of the 15 problems, and two of these are better than the best previously found by any method. Allowing the advanced start augmentation of TS-0 to run for the same duration, by using advanced starts from the 4 different seed solutions, does even better, finding better solutions than TS-0 in 3 cases and a worse solution in 1 case.

Several avenues are suggested for future research. First, we anticipate the value of identifying an "indicator" that signals which critical solutions are more likely than others to be useful as a basis for generating trial solutions. An appropriate threshold defined in terms of this indicator (which becomes refined as the search yields additional information), may allow trial solutions to be generated only for a special subset of critical events, therefore significantly reducing the computation time of the variants that create such solutions. Second, the development of such an indicator and threshold may lead to an altered design of the search process (for example, by inducing oscillations of a somewhat different pattern than those currently employed). Third, the question arises whether related information may be used simply to specify a progressively changing set of seed solutions for a collection of "re-starts." Conceivably, the structure of the method may beneficially change by employing a special approach to generate diverse seeds, which shifts the method to rely more fully on a restarting component. Since the design we use causes each starting solution (e.g., produced by Method A or PA) to be a local optimum, this approach can potentially be

stronger than the type based on the conception of a starting solution that may be far removed from local optimality.

Finally, we note again that no effort has been made to tune any parameters of our basic TS-0 method, on which all our current variants are based. The parameter values are the same as those used in our earlier study that examined smaller and easier problems. Reasonably, the parameters should be a function of problem size and adaptively determined, rather than pre-set, according to successes in finding improved solutions as the values are varied during the search. (E.g., changes in amplitude of the oscillation can be governed in this way.) In addition, our use of longer term memory and its integration with short term memory are very primitive, offering another source of potential improvement. We note that a recent advance that uses partially overlapping ideas in an evolutionary framework is provided by Lodi, Allemand and Liebling [10].

The types of gains that can be achieved simply by an adaptive setting of parameter values is illustrated by two additional runs we made on the "g" problems using TS-0, subsequent to making the runs summarized in Table 7.8. For the first set of runs, the parameters t, $MaxSpan$, and $MaxNewAssign$ were set to 7, 7, and 5. For the second set of runs, they were set to 7, 7, and 7. For each set of runs, each problem was run for a total of 400 cycles.

Each set of runs produced two new "best known" solutions. Run set #1 gave new best solutions for problems g10 and g14 while run set #2 gave best known solutions for problems g7 and g8. Collecting the best results from all methods considered in this paper, we have generated the following "best known" solutions for the "g" problems: (gl, 131456), (g2, 172788), (g3, 192565), (g4, 215679), (g5, 242367), (g6, 243293), (g7, 253590), (g8, 264268), (g9, 262658), (g10, 274375), (g11, 131631), (g12, 173615), (g13, 192864), (g14, 216168), (g15, 242832). The results from these additional runs motivate our interest in conducting a rigorous target analysis in order to better understand which parameter values are best suited for certain types of problems. Such analysis is currently underway.

Although those considerations indicate that latitude exists to improve our basic approach, the significant advances achieved by the current study over previous results may have useful implications in view of the wide range of problems that can be cast in the nonlinear QP formulation. In particular, the ability to obtain high quality solutions for 1000 variable QP problems with only one to four minutes of computing time on a PC opens the door to handling practical problems of these and larger sizes that have been beyond the scope of previous investigations in this area.

References

[1] Alidaee, B., G. Kochenberger, and A. Ahmadian (1994). 0-1 Quadratic Programming Approach for the Optimal Solution of Two Scheduling Problems, Int. J. Syst. Sci., 25, 401-408.

[2] Barahona, F.M. Jünger, and G. Reinelt (1989). Experiments in Quadratic 0-1 Programming, Math. Program. 44, 127-137.

[3] Chardaire, P., and A. Sutter (1995). A Decomposition Method for Quadratic Zero-One Programming, Management Sci. 41, 704-712.

[4] Gallo, G., P. Hammer, and B. Simone (1980). Quadratic Knapsack Problems, Math. Program. 12, 132-149.

[5] Glover, F., and G. Kochenberger (1996). Critical Event Tabu Search for Multidimensional Knapsack Problems. Meta Heuristics: Theory & Applications, Kluwer, 407-427.

[6] Glover, F., G. A. Kochenberger, and B. Alidaee (1998). Adaptive Memory Tabu Search for Binary Quadratic Programs, Management Sci 44, 336-345.

[7] Glover, F., and M. Laguna (1997). Tabu Search, Kluwer.

[8] Glover, F., M. Laguna, E. Taillard, and D. de Werra, eds, (1993). Tabu Search, Special issue of the Ann. Oper. Res. 41, J. C. Baltzer.

[9] Gulati, V.P., S.K. Gupta, and A.K. Mittal (1984). Unconstrained Quadratic Bivalent Programming, European J. Oper. Res. 15, 121-125.

[10] Lodi, A., K. Allemand, and T.M. Liebling (1997). A Genetic Algorithm for Quadratic 0-1 Programming, Working Paper, University of Bologna.

[11] Pardalos, P.M., and G.P. Rodgers (1988). Computational Aspects of a Branch and Bound Algorithm for Quadratic 0-1 Programming, Technical Report CS-88-04, Penn State University.

[12] Pardalos, P.M., and G.P. Rodgers (1989). Computational Aspects of Branch and Bound Algorithm for Quadratic 0-1 Programming, Computing, 45, 131-144.