Adaptive Memory Tabu Search for Binary Quadratic Programs

Fred Glover; Gary A. Kochenberger; Bahram Alidaee

Management Science, Vol. 44, No. 3. (Mar., 1998), pp. 336-345.

Stable URL:
http://links.jstor.org/sici?sici=0025-1909%28199803%2944%3A3%3C336%3AAMTSFB%3E2.0.CO%3B2-H

Management Science is currently published by INFORMS.

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at http://www.jstor.org/about/terms.html. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at http://www.jstor.org/journals/informs.html.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

JSTOR is an independent not-for-profit organization dedicated to creating and preserving a digital archive of scholarly journals. For more information regarding JSTOR, please contact support@jstor.org.

http://www.jstor.org/
Fri Sep 15 17:03:56 2006

Adaptive Memory Tabu Search for Binary Quadratic Programs

Fred Glover • Gary A. Kochenberger • Bahram Alidaee
Graduate School of Business, University of Colorado at Boulder, Boulder, Colorado 80309
College of Business, University of Colorado at Denver, Denver, Colorado 80217-3364
College of Business, University of Mississippi, University, Mississippi 38677

Recent studies have demonstrated the effectiveness of applying adaptive memory tabu search procedures to combinatorial optimization problems. In this paper we describe the development and use of such an approach to solve binary quadratic programs. Computational experience is reported, showing that the approach optimally solves the most difficult problems reported in the literature. For challenging problems of limited size, which are capable of being approached by exact procedures, we find optimal solutions considerably faster than the best reported exact method. Moreover, we demonstrate that our approach is significantly more efficient and yields better solutions than the best heuristic method reported to date. Finally, we give outcomes for larger problems that are considerably more challenging than any currently reported in the literature.
(*Integer Programming; Heuristics; Nonlinear Optimization*)

1. Introduction

Tabu search (TS) has been demonstrated to be a powerful solution methodology for a wide variety of difficult combinatorial optimization problems. Illustrative applications along with fundamental TS principles can be found in recent survey papers (Glover 1994, Glover and Laguna 1993, Glover et al. 1994). The distinguishing feature of TS is its use of adaptive memory structures coupled with associated strategies for exploiting information gathered during the search process. This adaptive memory component of TS typically incorporates both recency-based and frequency-based memory defined over varying short-term and long-term spans of the search history. The memory operates through control mechanisms of tabu restrictions and aspiration criteria, and associated penalties and inducements to modify move evaluations.

These constructs, implemented within a framework that fosters an effective interplay between intensification and diversification, have proved very successful in orchestrating a robust search of the solution space. Such an approach designed to solve multidimensional knapsack problems, recently reported in Glover and Kochenberger (1995), incorporates a "critical event" memory structure that we embody in our present investigation.

In this paper we present a tabu search implementation designed to solve the binary quadratic program:

$$\text{QP:} \quad \max x_0 = xAx,$$

where x is a zero–one vector.

Binary quadratic programs with an explicit linear component in the objective can always be put in this form, and thus we adopt this representation without loss of generality.

A large number of important applications of this model, which is known to be NP-hard, have been reported in the literature. For example, the QP problem has applications to capital budgeting and financial analysis problems (Laughunn 1970, McBride and Yormark 1980), CAD problems (Krarup and Pruzan 1978), traffic message management problems (Gallo et al. 1980, Witsgall 1975), and machine scheduling problems (Alidaee et al. 1994, Lakshminarayan et al. 1979). The QP for-

0025-1909/98/4403/0336$05.00
Copyright © 1998, Institute for Operations Research and the Management Sciences

mulation also encompasses many classical combinatorial optimization problems such as the maximum cut and maximum clique problems (Hammer and Simone 1987, Hammer et al. 1968, Pardalos and Phillips 1990), as well as a variety of nonlinear 0-1 programs.

Our approach is motivated by the success of recently developed tabu search strategies for multidimensional knapsack problems (Glover and Kochenberger 1995). As will be shown, an adaptation of these strategies yields a fruitful approach for QP problems as well. In the sections that follow, we discuss the basic TS ideas employed, outline their algorithmic implementation, discuss our computational experience, and conclude with some final remarks.

2. Basic Notions

The TS approach taken here is based on strategic oscillation, which is a nonmonotonic guidance mechanism that directly or indirectly underlies many tabu search implementations. Our procedure embodies a straightforward type of oscillation which alternates between constructive phases (progressively setting variables to 1) and destructive phases (progressively setting variables to 0). The process of setting variables to 1 during a constructive phase will be referred to as "adding variables" and the process of setting variables to 0 during a destructive phase will be referred to 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 to occur when the next move (either an add or a drop) causes the objective function to decrease. This definition is based on the fact that the search process is structured so that typically the most recent previous move before such a critical move causes the objective function to increase or remain unchanged. The solution corresponding to a critical event 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 variables added during a constructive phase or the number of variables dropped during a destructive phase. In this particular application, we maintain the same depth ("amplitude") of oscillation in both phases.

The parameter span itself is made to vary to compound the non-monotonic pattern of oscillation. The manipulation of the parameter span may be viewed as an outer oscillation that contains the oscillations of constructive and destructive phases within it.

We will not attempt to give a detailed picture of tabu search in this paper, but will focus the special details that underlie the present implementation, considering a particular subset of the strategies of tabu search that we find to be effective in the context of the QP problem. We first give an overview of relevant parameter and memory structures, followed by the description of the method that incorporates them.

3. Parameter Functions and Memory Structures

3.1. The Role of Span

The span parameter controls the pattern of oscillation in the following way. During a constructive phase, variables are added until the span parameter indicates it is time to switch to the destructive phase. The search process then turns around and proceeds toward (and typically past) the next critical event by dropping variables. Eventually, we switch again to the constructive phase and start once again adding variables. Sitting above this oscillation pattern, following Glover and Kochenberger (1995), is a related pattern that we use to control the span parameter itself.

We begin with span equal to 1 and gradually increase it to some limiting value. 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 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.

3.2. Using Recency and Frequency Information

In order to influence the search by recency information, we record (in a circular list) the last t solutions obtained at critical events. In our implementation, t is a simple function of problem size, and takes values in the range from 3 to 12 in the problems we have tested. For the results reported later in this paper, t was set equal to 3.

Using this record of the last t critical solutions, which range from x(last) to x(last−(t−1)), we maintain a short-term recency vector (TABU_R) that is the sum of the last t solutions. That is, each time a new x(last) is identified, we set

$$\text{TABU_R} = \text{TABU_R} + x(\text{last}) - x(\text{last-}t).$$

Although we refer to TABU_R as a "recency" vector, its entries record the frequency that variables have been assigned the value 1 (in the t most recent critical solutions).

Long-term frequency information is captured for use in the search process by maintaining another vector (TABU_F) which is the sum of all critical solutions encountered to date, rather than the last t critical solutions. Additional vectors can be maintained to record frequencies of values assignments over other spans of time, or to maintain frequency values that are discounted by time (as by exponential smoothing), but we have found that these two simple memory structures suffice to give very good results in the present application.

We use these memory structures to influence the search process to head in new directions as we pursue 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.

To illustrate the underlying rationale, suppose we have just switched from the destructive phase to the constructive phase. Other things equal, we would like to choose the variable (to add next) that contributes the largest net increase to the objective function value, x_0. 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 TABU_R(j) = 0. That is, we may conceive the condition TABU_R(j) > 0 as implying that x_j is tabu. More generally, we seek to require that the first k variables added back (after turn around) to have TABU_R(j) = 0. Such a requirement may not be strictly possible. Consequently, we attach a penalty weight, PEN_R, to TABU_R(j) and create a penalty value PEN_R*TABU_R(j). This penalty value is subtracted from the x_0 evaluation of variable x_j to provide a penalized evaluation.

Long-term frequency information is included by subtracting another penalty term, PEN_F*TABU_F(j), from the evaluation of variable x_j. The positive weight, PEN_F, is scaled by the iteration count so that the penalty influence derived from long-term frequency information is small compared to the somewhat larger PEN_R weight of the short-term recency information. In this manner, long-term tabu information plays a subtle but useful role of breaking ties (or "near ties") that may otherwise occur via the use of short-term information only. For the results reported later in this paper, we set PEN_R equal to the maximum element in the A matrix and PEN_F equal to PEN_R divided by 1000 times the number of transfer phase executions (iterations).

Similarly, when we switch from the constructive phase to the destructive phase, we want to restrict the choice of variables to drop so that the first k variables chosen are selected from those x_j with the maximum entries in the tabu lists TABU_R and TABU_F. Here, the tabu restriction is implemented by adding the short and long term penalty terms to the normal x_0 calculation and choosing the variable with the maximum evaluation. These calculations are summarized below.

Let eval(j) denote the evaluation of variable j and del $x_0(j)$ denote the net increase in x_0 associated with flipping x_j. Let count_var be the number of variables chosen since the last turn-around and let j^* be the index of the variable to be chosen next. Then in the constructive phase, we choose j^* (from the set of all j such that $x_j = 0$) as follows:

$$\text{If count_var} > k, \text{ let eval}(j) = \text{del } x_0(j).$$

$$\text{If count_var} \leq k, \text{ let eval}(j) = \text{del } x_0(j) - \text{PEN_R*TABU_R}(j) - \text{PEN_F*TABU_F}(j).$$

Then j^* corresponds to the variable with the maximum value of eval(j). In the destructive phase, we proceed as above (again to maximize) except that we *add* the penalty terms and search for j^* among the set of j such that $x_j = 1$.

The parameter k, which identifies the number of tabu-influenced adds or drops made immediately after a *turn-around*, is managed in a fashion that fosters addi-

tional diversity in the search process. We start with $k = 1$, and after a number of iterations (e.g., $2t$), we set $k = k + 1$. We continue in this fashion until k reaches a limit (KMAX), at which point we set k back to 1 and the process repeats. In general, KMAX should be a function of problem size. However, over a rather wide range of problems, KMAX in the range from 2 to 4 has performed well in our testing.

4. Overview of Method

Given the preceding discussion, we now present a general sketch of our method. Below we denote by $J(0)$ the set of variables currently equal to 0, $J(1)$, the set of variables currently equal to 1, x_0 the objective function value, and x^* the best solution found so far. The notation $x_0{:}(x_j = v)$ for $v = 0$ or 1, refers to the value of x_0 that results by changing the value of x_j from $1 - v$ to v, while holding the value of all other variables unchanged.

Initialization:

set $x = x^* = 0$, $x_0^* = -\infty$, count_span=0, k=1

while iter count (or # span cycles) < limit **do**

Constructive Phase:

{**comment:** adding variables up to critical event}

while $|J(0)| > 0$ **do**

find $j^* = \operatorname{argmax}\{x_0{:}(x_j = 1)\}$, $j \in J(0)$

if $x_0{:}(x_j^* = 1) > x_0$ **then**

set $x_j^* = 1$

else

if $xAx > x^*Ax^*$ **then**

set $x^* = x$

endif

endif

endwhile

{**comment:** add variables beyond critical event}

while count_span ≤ span and $|J(0)| > 0$ **do**

set count_span = count_span + 1

find $j^* = \operatorname{argmax}\{x_0{:}(x_j = 1)\}$, $j \in J(0)$

set $x_j^* = 1$

endwhile

Transfer Phase:

{**comment:** manage span (as specified later), do bookkeeping, etc.}

set count_span = 0

Destructive Phase:

{**comment:** dropping variables up to critical event}

while $|J(1)| > 0$ **do**

find $j^* = \operatorname{argmax}\{x_0{:}(x_j = 0)\}$, $j \in J(1)$

if $x_0{:}(x_j^* = 0) > x_0$ **then**

set $x_j^* = 0$

else

if $xAx > x^*Ax^*$ **then**

set $x^* = x$

endif

endif

endwhile

{**comment:** dropping variables beyond critical event}

while count_span ≤ span and $|J(1)| > 0$ **do**

set count_span = count_span + 1

find $j^* = \operatorname{argmax}\{x_0{:}(x_j = 0)\}$, $j \in J(1)$

set $x_j^* = 0$

endwhile

Transfer Phase:

{**comment:** manage span, do bookkeeping, etc.}

set count_span = count_span + 1

endwhile

The preceding method terminates whenever a selected number of iterations have been performed or after a preset number of complete span cycles have been performed.

4.1. Managing the Outer Oscillation Parameter (Span)

The oscillations about critical events are shaped by the parameter span. Span is fixed for a certain number of iterations and then changed in a systematic fashion. In our implementation, we manage span in the Transfer Phase by the following rules:

{**comment:** iter_span is the number of transfer phase executions at the current span value; dir denotes whether span is increasing or decreasing}

```
enter with values for span, p1, p2, iter_span, and dir
iter_span = iter_span + 1
if dir is increasing then
  if span ≤ p1 then
    if iter_span > p2*span then
      set span = span +1
      set iter_span = 0
    endif
  else
    if iter_span > 4 then
      set span = span +1
      set iter_span = 0
    endif
  endif
  if span > p2 then
    set span = p2
    set dir to decreasing
  endif
else
  if p1 +1 ≤ span ≤ p2 then
    if iter_span > 4 then
      set span = span − 1
      set iter_span = 0
    endif
  else
    if iter_span > p2*span then
      set span = span − 1
      set iter_span = 0
    endif
  endif
  if span = 0 then
    set span = 1
    set direction increasing
  endif
endif
```

Large values of $p1$ and $p2$ foster aggressive diversification, while smaller values facilitate a search in a more compact neighborhood around the most recent critical solution. The default values we employ are $p1 = 3$ and $p2 = 7$. These particular values have proven to be effective in a variety of problem settings including labor scheduling problems and multidimensional knapsack problems. As shown in the next section, they also worked well on the quadratic binary programs addressed in this paper.

5. Computational Experience

5.1. Initial Testing and Comparisons

Due to the widespread application as well as computational challenge of QPs, many researchers have reported algorithms designed to solve 0–1 quadratic programs over the years. A survey of early work is found in Hansen (1979). More recently, a variety of additional exact and heuristic approaches have been published and tested (Barahona et al. 1989, Chardaire and Sutter 1995, Gulati et al. 1984, Hammer and Simone 1987, Pardalos and Rodgers 1989, Williams 1985). Of the heuristic approaches in the literature, the decomposition method of Chardaire and Sutter (1995) recently published in *Management Science* is reported to be very successful on standard test problems. The most successful exact algorithm is the gradient-based branch and bound approach of Pardalos and Rodgers (1989). Despite the advances reported in the literature QP problems are, in general, difficult to solve. Later in this section, we offer a comparison of our approach with the two leading methods of Chardaire and Sutter (1995) and Pardalos and Rodgers (1989).

To test our approach, we made use of a standard testbed of problems provided by Pardalos and Rodgers (P&R) (1989), which has been designed to incorporate various characteristics as a challenge to other researchers. (P&R specify problem characteristics, provide a random number generator, and specify the random number seed to enable alternative algorithms to be compared on the same test problems.) While some problems are harder than others, all are believed to be difficult. We specifically tested our method on three classes of problems that we arbitrarily refer to as problem sets a, b, and c. The "a" problems were generated as specified by P&R in what they labeled the "WBJR experiments." There are eight such problems, varying in size form 40 to 100 variables with all A matrix elements randomly generated between −100 and 100.

The "b" problems were generated according to the problems P&R labeled the "Gulati experiments." There are 10 such problems, ranging in size from 20 to 125 variables. Here, the off-diagonal elements of A were generated between 0 and 100, while the diagonal elements were between −63 and 0. Finally, the "c" problems are the seven "most difficult" problems given by

P&R, which are instances of the "Barahona experiments." For these problems, off-diagonal elements are generated between −50 and 50, while diagonal elements are between −100 and 100.

Table 1 presents the results we obtained on these problems. Each problem is identified as to class, size, and density of A matrix. Moreover, the random number seed used is given for each problem as well. Each of the 25 problems was run for a span cycle limit of 5 cycles on a Pentium PC. Across all 25 problems, the time required for the five span cycles ranged from 2 to 141 seconds with an average of 47 seconds. For each problem, we report the best solution found along with the conditions of the search at the time this solution was obtained. For instance, for problem 1a, the best solution was found in span cycle #1, with span = 3 and $k = 1$, yielding $x_0 = 3414$. We also note that this solution was found in less than 1 second and that it took 37 seconds to complete the 5 cycles.

Our method found optimal solutions to all 25 problems. This was verified by solving each problem by the exact algorithm (Q01D.FOR) provided by P&R. The last two columns of Table 1 indicate, respectively, the time required for the branch-and-bound code to locate the optimal solution and to terminate, having proven optimality.

It is interesting to note that on 22 of the 25 problems, the optimal solution was found in the first span cycle. One problem took two span cycles, and two problems took three cycles. Thus the limit of five span cycles was

Table 1 Initial Testing

					Tabu Search Conditions at "Best" Solution					P&R	
ID	*n*	Den	Seed	Opt Obj Fun	Cycle #	SPAN	*k*	Time to (sec)	Time for 5 Cycles	Time to Opt	Time to Terminate
1a	50	0.1	10	3,414	1	3	1	<1	37	<1	<1
2a	60	0.1	10	6,063	1	1	1	<1	60	<1	<1
3a	70	0.1	10	6,037	1	2	2	15	71	32	133
4a	80	0.1	10	8,598	1	3	1	14	93	<1	219
5a	50	0.2	10	5,737	1	3	1	7	38	2	29
6a	30	0.4	10	3,980	1	2	1	<1	15	<1	3
7a	30	0.5	10	4,541	1	1	1	<1	14	<1	4
8a	100	0.0625	10	11,109	1	3	1	17	141	<1	2
1b	20	1.0	10	133	1	1	1	<1	2	<1	<1
2b	30	1.0	10	121	1	6	2	<1	4	<1	<1
3b	40	1.0	10	118	1	5	1	<1	5	<1	<1
4b	50	1.0	10	129	1	1	1	<1	8	<1	<1
5b	60	1.0	10	150	1	1	1	<1	11	<1	1
6b	70	1.0	10	146	1	3	1	<1	14	<1	1
7b	80	1.0	10	160	1	1	1	<1	18	<1	2
8b	90	1.0	10	145	1	6	3	5	21	<1	3
9b	100	1.0	10	137	2	2	1	11	27	<1	5
10b	125	1.0	10	154	1	5	2	5	46	1	8
1c	40	0.8	10	5,058	1	1	1	<1	27	<1	1,805
2c	50	0.6	70	6,213	1	1	1	<1	40	<1	30,400
3c	60	0.4	31	6,665	1	3	2	3	64	<1	*
4c	70	0.3	34	7,398	1	6	1	10	74	<1	18,003
5c	80	0.2	8	7,362	3	2	1	41	100	983	16,051
6c	90	0.1	80	5,824	3	2	1	49	111	43	160
7c	100	0.1	142	7,225	1	3	1	7	140	230	248

* Did not terminate in 17 hours. However, the solution given was proven by Pardalos and Rodgers (1989) to be optimal (private correspondence).

more than adequate. Across all problems, there is some variation in the value of the parameter span at which the best solution was obtained. Most often, however, the best solution was found at span = 3. Values of k most often associated with the best solution were 1 and 2, with $k = 1$ occurring most frequently.

Note that the exact algorithm of P&R performed exceptionally well on most of the problems in Table 1. For the easily solved "a" and "b" problems, it recorded shorter computational times than our tabu search heuristic. Such performance appears to owe significantly to the ingenuity of their approaches for generating starting solutions, since in most cases these first solutions turned out to be optimal for these simple problems.

The performance of the P&R approach on the more challenging "c" problems, however, is quite different. Their starting points were still good, but less frequently optimal, and their method generally took excessive times to prove optimality for these problems. In general, larger size and greater density degrades the performance of the P&R algorithm. This degradation is increased when the distribution of the gradient range is fairly uniform. Pardalos and Rodgers (1989, p. 143) purposely generated the "c" problems with these characteristics to illustrate the difficulty such problems pose for their gradient based branch and bound approach. The results reported in Table 1, consistent with those reported in by P&R in Pardalos and Rodgers (1989), bear out this difficulty.

In contrast to the P&R results, note that the performance of our tabu search heuristic is quite uniform across all 25 problems in Table 1—including the "c" problems. In all cases, we found optimal solutions in a few span cycles and within modest computational times. This is somewhat surprising, from the point of view of the low heuristic content of our choice rules, since we did not attempt to incorporate any particular ingenuity into these rules, but relied primarily on the searching power of the critical event memory.

The "c" problems of Table 1 illustrate that for certain modest sized problems, the exact method of P&R runs into severe practical limitations. To further explore the limitations of the P&R algorithm on modest sized (but difficult) problems, and to provide additional comparisons with our tabu search heuristic, we generated 10 additional test problems of size = 100 with densities ranging from 0.1 to 1.0. For each of these problems, the off-diagonal elements were randomly generated between ±50, and the diagonal elements were taken to be between ±75. The results of our testing are shown in Table 2.

These 10 problems, with characteristics similar to the "c" problems, proved to be beyond the practical capability of the P&R algorithm. Each problem was initially run for a limit of five span cycles on our tabu search heuristic and a limit of five million vertices on the P&R branch-and-bound algorithm. Table 2 reports the best solution found by each approach and the corresponding computational times in seconds. For only one of the problems (1d), did the branch and bound algorithm match our TS approach in terms of solution quality. In all cases, the algorithm of P&R took considerably more time (generally about 50 times longer) in spite of failing to obtain solutions of matching quality.

Subsequent runs of the branch and bound algorithm were made on all 10 problems with the default limit (given by P&R) of 134 million vertices. In no case did the algorithm terminate with a completed tree search before the default limit was reached. Moreover, in no case did they improve on the solutions reported in Table 2. On the Pentium 90 PC, these runs took more than 30 hours each.

Table 2, consistent with the performance reported in Table 1, shows that our approach efficiently generated attractive solutions to each of these problems. In all cases but one, where the two methods found the same solution, our approach generated solutions superior to those found by the P&R procedure while investing only a small fraction of the computation time. For each problem, the time required for the five span cycles was roughly one minute, illustrating both the efficiency and robustness of our approach.

The more recent heuristic method of Chardaire and Sutter (C&S) reported in *Management Science* (Chardaire and Sutter 1995) is also applied to a subset of the P&R test problems. (Their method is not applied to larger problems, however, such as those indicated subsequently.) C&S report testing on selected problems from the "a" problem category with varying densities ranging in size from 40 to 100 variables. Average performance is cited over ten problems for each size and density. For small to medium sized problems (50 variables

Table 2 Additional 100 Variable Problems

			Tabu Search			P&R	
ID	Den	Seed	Best Obj Fun	Time to Best	Time for 5 Cycles	Best Obj Fun	Time for 5 Million Vertices
1d	0.1	31	6,333	3	60	6,333	3,885
2d	0.2	37	6,579	18	56	6,467	3,242
3d	0.3	143	9,261	3	62	9,205	3,343
4d	0.4	47	10,727	7	59	10,705	3,203
5d	0.5	31	11,626	53	64	11,589	3,288
6d	0.6	47	14,207	32	64	14,017	3,425
7d	0.7	97	14,476	23	63	13,999	3,441
8d	0.8	133	16,352	11	63	16,143	3,503
9d	0.9	307	15,656	1	58	15,584	3,507
10d	1.0	1,311	19,102	2	70	18,930	3,504

Note: 1. All problems are of size $n = 100$ with off diagonal elements between +/− 50 and diagonal elements between +/− 75. All times are in seconds on a Pentium 90 computer.

2. For all 10 problems, the initial P&R solutions (which were found in less than 1 second) were not improved upon or verified as optimal in five million vertices generated by their branch and bound algorithm.

or less), C&S typically find an optimal solution. However, for larger problems, even of this relatively simple type, their solution quality falls off and enlarged gaps appear between the best lower and upper bounds they compute on the optimal objective function value. Chardaire and Sutter comment (1993, p. 71) ". . . our method has some limits. Gaps increase with size of problem."

For problems of size $n = 75$, C&S failed to find the optimal solution on 21 of the 30 problems attempted. For problems of size $n = 100$, the optimal solution was found for only one of the 30 problems attempted. No experience was reported for problems larger than $n = 100$. In contrast, we should note that P&R report readily finding optimal solutions to low density problems from this category with up to 100 variables. And, as discussed previously, our tabu search approach had no difficulty finding optimal solutions to all such problems.

In addition to the degradation in solution quality with problem size, the solution times reported by C&S seem to be quite large compared to our experience on similar problems. For problems of size $n = 75$, they report average solution times as large as 12 minutes (HP 720 workstation) and, for problems of size $n = 100$, they report average times as large as 45 minutes. Considering both solution quality and solution time, it appears that their approach is suitable only for small problems (50 variables or less). For larger problems, our TS approach gives better performance on both measures.

5.2. Additional Testing

In an effort to test our approach on larger problems, we generated 10 additional problems with the characteristics of the "c" problems. Five of these problems are for $n = 200$, and five are for $n = 500$. Due to their size and characteristics, these problems should be very difficult to solve. In fact, we believe them to be the most challenging problems reported in the literature to date—far beyond the capabilities of current exact methods and challenging as well for heuristic approaches. Our results are shown in Table 3.

Each problem was run for an arbitrary preset limit of outer span cycles. For the 200 variable "e" problems, the limit was set at 10 cycles, and the times reported in the table are seconds on a Pentium 90 PC. For the 500 variable "f" problems, the span limit was set at 20 cycles, and the times reported are in seconds on a VAX Alpha 2100 model 300 computer. Note that these larger problems often took several span cycles to locate the

Table 3 Larger Test Problems

Problem	n	Density	Seed	Search Condition at "Best" Solution: Objective Function	Cycle #	Span	Iter	k	Time (sec)	Time for Cycle Limit
1e	200	0.1	51	16,464	5	3	598	1	374	876
2e	200	0.2	43	23,395	2	7	207	1	149	913
3e	200	0.3	34	25,243	1	7	63	3	56	1,069
4e	200	0.4	73	35,594	1	2	20	1	21	1,054
5e	200	0.5	89	35,154	3	6	362	1	284	1,007
1f	500	0.10	137	61,194	10	1	1,399	2	506	985
2f	500	0.25	137	100,161	4	3	528	1	211	1,001
3f	500	0.50	137	138,035	1	7	69	2	41	980
4f	500	0.75	137	172,771	11	3	1,440	1	504	967
5f	500	1.00	137	190,507	3	3	309	1	118	1,018

Note: 1. These problems were generated according to the approach given by P&R as used for the "c" problems of Table 1. The termination limit for the "e" problems was arbitrarily set at 10 span cycles. For the "f" problems, the termination limit was set at 20 span cycles for each problem.

2. The "e" problems were generated with off diagonal elements between +/− 50 and the diagonal elements between +/− 100. The times given for the "e" problems are in seconds on a Pentium 90 PC.

3. The "f" problems were generated with off diagonal elements between +/− 50 and the diagonal elements between +/− 75. Due to the size and complexity of these problems, they were run on a VAX Alpha 2100 model 300 computer. This machine is roughly five times faster than the Pentium 90. Times shown are in seconds.

best solution found. Nonetheless, our approach quickly located solutions to these larger problems. Here, however, we have no proof of optimality, since these problems are beyond the scope of those that can be handled within practical time limits by exact algorithms.

6. Conclusions

We report a new tabu search approach to binary quadratic programs, based on a flexible memory system that utilizes recency and frequency information from critical events encountered during the search process. The method incorporates a strategic oscillation scheme that alternates between constructive and destructive phases, and drives the search to variable depths on each side of critical solutions.

Our approach successfully (and quickly) found optimal solutions for test problems characterized as challenging in previous studies. On problems representing a nontrivial level of difficulty, our approach significantly outperformed both the best exact method and the best heuristic method previously reported in the literature. Moreover, it quickly found good (perhaps optimal) solutions to difficult problems larger than those examined in previous investigations—problems too large to permit optimality to be verified by exact methods that represent the state of the art. These results were obtained using a straightforward implementation of our tabu search procedure with a simple choice rule and a minimum of parameter experimentation. This success, coupled with the results reported in Glover and Kochenberger (1995) on multidimensional knapsack problems, highlights the effectiveness of critical event memory schemes in tabu search and suggests they may find useful applications to solving other types of optimization problems.

References

Alidaee, B., G. Kochenberger, and A. Ahmadian, "0-1 Quadratic Programming Approach for the Optimal Solution of Two Scheduling Problems," *International J. Systems Sci.*, 25 (1994), 401–408.

Barahona, F., M. Junger, and G. Reinelt, "Experiments in Quadratic 0-1 Programming," *Math. Programming*, 44 (1989), 127–137.

Chardaire, P. and A. Sutter, "A Decomposition Method for Quadratic Zero-One Programming," *Management Sci.*, 41, 4 (1995), 704–712.

Gallo, G., P. Hammer, and B. Simone, "Quadratic Knapsack Problems," *Math. Programming*, 12 (1980), 132–149.

Glover, F., "Tabu Search Fundamentals and Uses," Working Paper, University of Colorado, Denver, CO, 1994.

Glover, F. and G. A. Kochenberger, "Critical Event Tabu Search for Multidimensional Knapsack Problems," Working Paper, University of Colorado, Denver, CO, 1995.

—— and M. Laguna, "Tabu Search," in C. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publishing, 1993, 70–141.

——, ——, E. Taillard, and D. de Werra (Eds.), "Tabu Search," Special Issue of the *Ann. Oper. Res.*, 41, (1994).

Gulati, V. P., S. K. Gupta, and A. K. Mittal, "Unconstrained Quadratic Bivalent Programming," *European J. Operations Res.*, 15 (1984), 121–125.

Hammer, P. and B. Simone, "Quadratic Functions of Binary Variables," Technical Report RRR #20-87, RUTCOR, Rutgers University, New Brunswick, NJ, 1987.

——, P. Hansen, and B. Simone, *Boolean Methods in Operations Research*, Springer, New York, 1968.

Hansen, P., "Methods of Nonlinear Programming," *Ann. Discrete Math.*, 5 (1979), 52–70.

Krarup, J. and P. A. Pruzan, "Computer Aided Layout Design," *Math. Programming Study*, 9 (1978), 75–94.

Lakshminarayan, S., R. Lakshmanan, R. Papineau, and R. Rochette, "Order Preserving Allocation of Jobs to Two Non-Identical Parallel Machines: A Solvable Case of the Maximum Cut Problems," *INFOR*, 17 (1979), 230–241.

Laughunn, D. J., "Quadratic Binary Programming," *Operations Res.*, 14 (1970), 454–461.

McBride, R. D. and J. S. Yormark, "An Implicit Enumeration Algorithm for Quadratic Integer Programming," *Management Sci.*, 26 (1980), 282–296.

Pardalos, P. M. and A. T. Phillips, "A Global Optimization Approach for Solving the Maximum Clique Problem," *International J. Computational Math*, 33 (1990), 209–216.

—— and G. P. Rodgers, "Computational Aspects of a Branch and Bound Algorithm for Quadratic 0-1 Programming," *Computing*, 45 (1989), 131–144.

Williams, A. C., "Quadratic 0-1 Programming Using the Roof Dual with Computational Results," RUTCOR Research Report C#8-85, Rutgers University, New Brunswick, NJ, 1985.

Witsgall, C., "Mathematical Methods of Site Selection for Electric System (EMS)," NBS Internal Report, 1975.

Accepted by T. M. Liebling; received March 1996. This paper has been with the authors 3 months for 1 revision.