ELSEVIER

European Journal of Operational Research 137 (2002) 272–287

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

One-pass heuristics for large-scale unconstrained binary quadratic problems

Fred Glover *, Bahram Alidaee, César Rego, Gary Kochenberger

Hearin Center for Enterprise Science, University of Mississippi, University, MS 38677, USA

Abstract

Many significant advances have been made in recent years for solving unconstrained binary quadratic programs (UQP). As a result, the size of problem instances that can be efficiently solved has grown from a hundred or so variables a few years ago to 2000 or 3000 variables today. These advances have motivated new applications of the model which, in turn, have created the need to solve even larger problems. In response to this need, we introduce several new "one-pass" heuristics for solving very large versions of this problem. Our computational experience on problems of up to 9000 variables indicates that these methods are both efficient and effective for very large problems. The significance of problems of this size is that they not only open the door to solving a much wider array of real world problems, but also that the standard linear mixed integer formulations of the nonlinear models involve over 40,000,000 variables and three times that many constraints. Our approaches can be used as stand-alone solution methods, or they can serve as procedures for quickly generating high quality starting points for other, more sophisticated methods.

Keywords: Unconstrained binary quadratic optimization; One-pass heuristics

1. Introduction

The unconstrained quadratic program can be written in the form:

$$\text{UQP}: \quad \min f(x) = xQx,$$

where Q is an $n \times n$ matrix of constants and x is an n-vector of binary variables. UQP is notable for its ability to represent a wide variety of important problems as well as its NP-hard difficulty. Applications have been reported in many different settings including social psychology [14], financial analysis [18,20], computer aided design [17], traffic management [7,27], machine scheduling [1], cellular radio channel allocation [6], and molecular conformation [25]. Moreover, many combinatorial optimization problems pertaining to graphs such as determining maximum cliques, maximum cuts, maximum vertex packing, minimum coverings,

* Corresponding author.

E-mail addresses: fglover@bus.olemiss.edu (F. Glover), balidaee@bus.olemiss.edu (B. Alidaee), crego@bus.olemiss.edu (C. Rego), gkochenberger@bus.olemiss.edu (G. Kochenberger).

0377-2217/02/$ - see front matter
PII: S0377-2217(01)00209-0

maximum independent sets, and maximum independent weighted sets are known to be capable of being formulated by the UQP problem (see, for example, the surveys of Pardalos and Rodgers [23], and Pardalos and Xue [24]).

The application potential of UQP is even much greater than this, however, due to reformulation methods that enable certain constrained models to be re-cast in the form of UQP. Hammer and Rudeanu [13] show that any quadratic (or linear) objective in bounded integer variables and constrained by linear equations can be reformulated as a UQP model. Such reformulations have been recently highlighted by Kochenberger et al. [16], where several classical models are examined, reformulated, and solved as unconstrained quadratic programs. This reformulation approach was successfully used to solve quadratic knapsack problem as recently reported by Glover et al. [10]. Considering such reformulation possibilities, the UQP model has a vast range of applications.

Due to its computational challenge and application potential, UQP has been the focus of a considerable number of research studies in recent years, including both exact and heuristic solution approaches. Notable recent studies addressing UQP are those by Williams [26], Pardalos and Rodgers [23], Boros et al. [5], Chardaire and Sutter [6], Glover et al. [9,11], Alkhamis et al. [2], Beasley [4], Lodi et al. [19], Amini et al. [3], and Glover et al. [8]. Other promising work is reported by Katayama et al. [15] and Merz and Freisleben [21]. These various studies approach the problem by branch and bound, decomposition, tabu search, simulated annealing, and evolutionary methods such as genetic algorithms and scatter search. Each of these approaches exhibits some degree of success. However, the exact methods degrade rapidly with problem size, and have meaningful application to general UQP problems with no more than 100 variables. For larger problems, heuristic methods are required. To date, tabu search and the evolutionary methods have proven to be successful on problems of up to 3500 variables. For problems approaching this size, however, the genetic algorithm methods degrade substantially with density of the Q matrix and both tabu search and scatter search degrade in terms of solution time in order to locate solutions of high quality. While current methods have greatly expanded the size of problems that can be handled reasonably well, they are, in their present state of development, inadequate for the task of solving problem instances representing many important applications, which can easily range three or more times larger than current problem size limits.

This paper, in the interest of quickly finding solutions to these larger problems, is concerned with the study of "simple" one-pass heuristics for large-scale 0–1 UQP problems. Such approaches can serve as "stand-alone" methods or as advanced starting point procedures for more sophisticated (and time consuming) methods.

The rest of the paper is organized as follows. In Section 2, we present the one-pass heuristics designed for this study. To provide a basis for comparison as well as a rationale for our heuristics, we begin with a discussion of DDT, the best known and most promising one-pass heuristic in the current literature. Then, in Section 3, we present our computational experience. Our computational work is divided into two parts. The first part provides a relative comparison of the various one-pass methods by extensive testing on new test problems ranging from 1000 to 9000 variables. The second part shows how the best of the methods perform on standard test problems where "best known" solutions are available. Finally, Section 4 presents conclusions and comments on future work.

2. DDT and alternative one-pass heuristics

We start from the basic formulation of the unconstrained quadratic programming problem (UQP), which as previously noted can be stated as

$$\text{UQP}: \quad \min\ xQx,$$

where Q is an $n \times n$ symmetric matrix and x is a vector of n binary variables. Since for each binary variable we have x_i^2 equal to x_i, the problem also can be written as

$$\text{UQP}: \quad \min \sum_{i=1}^{n} q_i x_i + \sum_{\substack{i=1 \\ i \neq j}}^{n} q_{ij} x_i x_j,$$

where q_i is the ith element of the diagonal of Q and q_{ij} is the (i,j)th element of Q. For a binary variable x_i let $\bar{x}_i$ be the complement of x_i, i.e., $\bar{x}_i = 1 - x_i$. Any element of the form x_i or $\bar{x}_i$ is called a *literal*. Let $T_j = u$ or uv for some literals u and v. Now, the UQP also can be expressed as

$$\text{UQP}: \quad \min \sum_j c_j T_j.$$

By an appropriate choice of literals (i.e., an appropriate choice of variables to complement in given terms), the coefficient c_j of T_j can be assumed positive. This is called a *posiform* expression of UQP [13,24]. There are typically many ways a UQP can be transformed to an equivalent *posiform* by substituting complements of some variables in UQP. The notion of a posiform plays a key role in the methods subsequently described.

Our discussion of one-pass heuristics can be put in perspective by reviewing the well-known DDT method of Boros et al. [5]. This algorithm, which is given in Fig. 1, is applied to a posiform representation of UQP. We use the framework of Fig. 1 to provide a point of departure for the development of the alternative one-pass heuristics. However, the computational results presented later for the DDT method follow the suggestion of Boros et al. [5] by using an implementation based on signed graphs. This method employs a special graph-based approach having computational efficiencies. (The reader is referred to [5] for the details.)

To illustrate, consider the following example from [5].

Example 1. We seek to minimize the quadratic function given by

$$Z = 13 - 5x_1 + 9x_2 + x_3 + 12x_4 + 7x_5 - 12x_1x_2 + 8x_1x_4 + 4x_2x_3 - 10x_2x_4 - 6x_3x_4 - 8x_4x_5.$$

An equivalent posiform is

$$Z = -10 + 17\bar{x}_1 + 12x_1\bar{x}_2 + 10x_2\bar{x}_4 + 8x_1x_4 + 8x_4\bar{x}_5 + 7x_5 + 6x_3\bar{x}_4 + 5\bar{x}_3 + 4x_2x_3 + 4x_4 + \bar{x}_2.$$

A step-by-step procedure of DDT applied to such posiform is as follows:

1. $T = \bar{x}_1,\ S = \bar{x}_1,\ \text{LC} = \{x_1 = 1\}$ and $Z = -10 + 13\bar{x}_2 + 12x_4 + 10x_2\bar{x}_4 + 8x_4\bar{x}_5 + 7x_5 + 6x_3\bar{x}_4 + 5\bar{x}_3 + 4x_2x_3$.
2. $T = \bar{x}_2,\ S = \bar{x}_1 \vee \bar{x}_2,\ \text{LC} = \{x_2 = 1\}$ and $Z = 4 + 8x_4\bar{x}_5 + 7x_5 + 6x_3\bar{x}_4 + 2x_4\bar{x}_5$.
3. $T = x_4\bar{x}_5,\ S = \bar{x}_1 \vee \bar{x}_2 \vee x_4\bar{x}_5,\ \text{LC} = \emptyset$ and $Z = 4 + 8x_4\bar{x}_5 + 7x_5 + 6x_3\bar{x}_4 + 2x_4 + \bar{x}_5$.
4. $T = x_5,\ S = \bar{x}_1 \vee \bar{x}_2 \vee x_4\bar{x}_5 \vee x_5,\ \text{LC} = \{x_4 = x_5 = 0\}$ and $Z = 5 + 5x_3$.
5. $T = x_3, S = \bar{x}_1 \vee \bar{x}_2 \vee x_4\bar{x}_5 \vee x_5 \vee x_3, \text{LC} = \{x_3 = 0\}$ and $Z = 5$, with $x = (1,1,0,0,0)$.

The DDT approach operates by implicitly indexing the terms of the posiform so that $c_i \geqslant c_j$ for $i < j$ (though we only need to pick up a current $\text{Max}(c_i)$ at each step). This procedure is reported to perform well on a wide variety of problems and to be particularly effective on problems of low density [5,12].

Initialization: Input UQP in a *posiform*, call it Z, set $S = 0$ and let L be the set of all elements of posiforms T
Devour: Find a term T from L with the largest c_j, let $L = L \setminus \{T\}$ and set $S = S \cup \{T\}$
Digest: Draw all the logical conclusions LC of the Boolean equation $\vee T = 0$, where $\vee$ is taken over all elements T in S
Tidy-up: Substitute the consequence LC, drawn in the previous phase, into Z, update LC, L and c_j for all T in L. If LC is empty then return to Devour.
Output: Solve the Boolean equation $\vee T = 0$ over all T in S, and output x. STOP.

Fig. 1. DDT Procedure.

2.1. Alternative one-pass heuristics

Taking DDT as a starting point, we create several alternative heuristics based on assumptions about the relative attractiveness of alternative choices that go beyond ranking the T_i terms in sequence. One set of variations additionally suggests alternative criteria are available for evaluating the choices given by the c_i values – or more precisely, that different ways of generating posiform representations (each of which creates a different set of c_i values) can be analyzed to yield new values to replace the c_i values, created as "hulls" or "composites" of the c_i values (or created by a special assignment strategy).

First, we discuss variants that retain the DDT policy of creating c_i values by complementing only the smaller index variable (or only the larger index variable) in each complementation step, but introducing assumptions for evaluating the terms that are slightly different than those used in the DDT procedure.

(A) The DDT approach often results in setting several variables to 0 or 1 simultaneously, as can be seen from Example 1. These assignments are triggered by giving a value to a literal that appeared in preceding pairs in the sequence. We conjecture that a more effective set of assignments will often result if only one of the variables in such a collection is given its implied assignment, followed immediately by updating the posiform representation before creating other assignments. While DDT performs very well on a wide variety of problems, we will nonetheless see in Section 3 that our computational results confirm the conjecture for a variety of problem instances.

Different ways of evaluating the contributions of the variables in the one-at-a-time assignments discussed in (A) lead to alternative ways of implementing the one-pass idea. Eight different evaluation schemes, accompanied by various implementation alternatives, are explained below. We denote the first literal encountered in the ordered sequence by u and denote the negation of u by $\bar{u}$.

(A1) Conceive that earlier terms have stronger "votes" for assignments. Consider the first term of the form $\bar{u}v$ which precedes the occurrence of u in the sequence. (That is, suppose $T_j = u$, and consider the smallest $i(< j)$ for which there is a term of the form $T_i = \bar{u}v$.) Given that the vote for $\bar{u}v$ is stronger than the vote for subsequent terms, and $\bar{u}$ is negated by the subsequent occurrence of u, then v inherits the strongest vote. Hence, the rule for this case is to make the assignment based on the literal v, i.e., by setting $v = 1$, and then updating before proceeding.

(A2) Conceive that a vote for a paired term is a "fuzzy" vote, which translates into a vote for each component of the pair, where these component votes are weaker than the vote for the pair itself. Since the subsequent literal u is the only basis for activating the consequence v (from the earlier term $\bar{u}v$), the literal itself is the most important assignment, and hence u receives the strongest vote. Thus the rule for this case is to make the assignment based on the literal u. This reasoning has added support described later.

(A3) To evaluate a given term T_i and its components more effectively, consider the two cases $T_i = u$ or $T_i = uv$ (for arbitrary u and v). In the first case, for $T_i = u$, let $\mathit{VoteStrength}(u) = F(c_i)$, where $F(c)$ is a monotonic increasing function of c. For example, let $F(c) = c$, or $F(c) = c^2$, etc. (Thus ordering by c is the same as ordering by $F(c)$.) In the second case, for $T_i = uv$, let $\mathit{VoteStrength}(u) = \mathit{VoteStrength}(v) = f \times F(c_i)$, where f is a fraction $\leqslant 0.5$. (The "fuzzier" or "weaker" the vote for each of u and v, the smaller the fraction.) Then $\mathit{TotalVote}(u)$ for each literal u is just the sum of all the $\mathit{VoteStrength}(u)$ values over the T_i terms, and $\mathit{NetVote}(u) = \mathit{TotalVote}(u) - \mathit{TotalVote}(\bar{u})$. These values can be determined by a single pass of the terms in the sequence, taking less work than actually ordering the T_i. (In a simple variant, the pass is restricted to the subsequence that terminates with the first term that is a literal. This takes a little more effort than going through the sequence, since it may be necessary to pick up several max terms before reaching the literal.) Thus, the rule is to make an assignment based on the literal u that receives the maximum $\mathit{NetVote}(u)$.

(B) There are additional reasons for diminishing the emphasis on using paired terms for evaluating the contributions of variables, suggesting it is more appropriate to give a small value to the

"fraction" term above. Specifically, the structure of the one-pass method as constituted so far assures that a number of possible valid implications can never be generated. To illustrate, consider the following sequence that implies $x_1 = 1$: (a) If $x_1 = 0$, then $x_2 = 0$; (b) if $x_2 = 0$, then $x_3 = 0$; (c) if $x_3 = 0$, then $x_1 = 1$. In terms of pairs, this corresponds to (a) $\bar{x}_1 x_2$; (b) $\bar{x}_2 x_3$; (c) $\bar{x}_1 \bar{x}_3$. But no terms T_i have the form of (c), so the implication cannot be created by reference to the type of posiform considered. On the other hand, the simple variation that interchanges $x_1 = 0$ and $x_1 = 1$ in this sequence avoids the appearance of a double negation, but still will not be generated by the current scheme. In this case the sequence is (a) $\bar{x}_1 x_2$; (b) $\bar{x}_2 x_3$; (c) $x_1 \bar{x}_3$. This can never be generated by a rule that adopts the convention of always choosing the smaller index variable or the larger index variable to be the one to complement.

Since the structure of the method can mask potentially valid implications from paired terms, the relevance of inferences based on these terms can appropriately be discounted. (In situations where such inferences should be needed, the approach of (A3) above is applicable.) Following this thread to the extreme, we consider a variant that disregards paired terms entirely, and merely picks the highest ranking literal at each step, which is effectively the strategy (A2). In this case, the process is accelerated, because the identification of the max term is restricted to looking at literals. If more than one literal has the same max value $c_i = c^*$, then ties can be broken using (A3), looking only at terms T_j such that $c_j \geqslant c^*$. (The tie-breaking slows the method slightly, but our later computational experience shows it can be useful. In this case, no decision needs to be made about the size of f since $f = 1$ is equivalent to any other positive value when the use is merely to break ties.)

To account for the fact that the c_i values are arbitrary, and may be replaced by other values in the 0–1 quadratic programming formulation, let q_i be the current updated coefficient of x_i, and q_{ij} be the current updated coefficient of $x_i x_j$. We disregard the order, and so take $q_{ij} = q_{ji}$. Let $N = \{1, \ldots, n\}$ be the index set for all current x_i (those left to be assigned values), and let $N(i) = \{j \in N - i : q_{ij} < 0\}$. Let $J(i)$ be the subset of $N(i)$ given by $J(i) = \{j : x_i x_j$ is *replaced by* $x_i(1 - x_j)\}$. Then we identify the value v_i given by: $v_i = q_i + \sum_{j \in J(i)} q_{ij}$, where $c_i = v_i$ if x_i is retained as a literal and $c_i = -v_i$ if x_i is complemented. Since v_i can take a range of values depending on the choice of $J(i)$, leading to different choices for a "best" assignment of values to variables, we consider the following variants:

(V1) Let $\text{vmax}_i = q_i$ and $\text{vmin}_i = q_i + \sum_{j \in N(i)} q_{ij}$. (If $N(i) = \emptyset$, then $\text{cmax}_i = \text{cmin}_i = q_i$.) Let c_{1i} and c_{2i} be the absolute values of vmax_i and vmin_i, respectively, and let $c_i^* = \text{Max}(c_{1i}, c_{2i})$. Then c_i^* is be the highest possible value of c_i over any method for choosing the sets $J(i)$. By a philosophy of choosing the "best of the best" we therefore select $i^* = \text{argmax}\{c_i^* : i \in N\}$ to give the assignment $x_i^* = 0$ or 1. (This choice is based on looking at the "outer hull" of the c_i values.)

(V2) Let $\text{cmean}_i = \{c_{1i} + c_{2i}\}/2$. This gives a more moderate evaluation than (V1) by choosing $i^* = \text{argmax}\{\text{cmean}_i\}$.

(V3) A "best of the worst" philosophy is implemented by defining $\text{cmean}_i = 0$ if $\text{vmax}_i > 0$ and $\text{vmin}_i < 0$, and otherwise defining $\text{cmin}_i = \text{Min}(c_{1i}, c_{2i})$. Then the choice is be to choose $i^* = \text{argmax}\{\text{cmin}_i\}$.

(V4) A "pairwise max" strategy consists of going through the relevant pairs (i, j), in any order as follows: (i) Start with $v_i = q_i$ for all i. (ii) As each (i, j) (with $q_{ij} < 0$) is encountered let $v_i = v_i + q_{ij}$ or $v_j = v_j + q_{ij}$ according to the choice that creates the current largest value of $\text{Max}(|v_i|, |v_j|)$. Finally, let c_i be determined as normally would be done from this assignment, and choose $i^* = \text{argmax}\{c_i\}$. (The order of examining the pairs (i, j) can make a difference, but we do not examine this effect.)

(V5) The same as (V4) but in step (ii) choose the change that creates the current largest value of $\text{Min}(|v_i|, |v_j|)$.

The preceding options can be implemented very quickly, by a single pass of the (i, j) values for $q_{ij} < 0$. Options V4 and V5 additionally can be used within strategies (A1) and (A3) above. All of these approaches can use (A3) for tie-breaking. In the following section we report implementations of several variations of the above algorithms.

3. Computational experiments

Our computational testing consists of two phases. Phase I provides a relative comparison of the one-pass methods. Our intent here is to determine if one or more of the methods stands out in competition with the others. For this purpose, we used a variety of randomly generated problems of various sizes and densities. Phase II is then carried out to see how the best of the one-pass methods performed on existing problems from the literature with known "best" solutions. Details of our testing are given below.

3.1. Phase I. Relative comparison of methods

To provide a test bed for judging the relative merits of the one-pass methods, we randomly generated a new set of test problems using the generator of Pardalos and Rodgers [23]. This generator is widely used by researchers in the area. As expected, the *difficulty* of the instances produced by this generator increases rapidly with both size and density. Our experience with exact methods (in previous work) applied to general instances produced by the generator is that exact methods are limited to problems with fewer than 150 variables. Heuristic methods are required to address larger problems – or even modest sized problems of high density.

The problems used in this study vary in density from 15% to 85% and vary in size from 1000 to 9000 variables. The data range was $[-20, 50]$ for off diagonal elements and $[-100, 100]$ for the diagonal elements of Q. For each combination of size and density, three instances are generated and solved, for a total of 168 problems in all. Assessments of performance in terms of solution quality and solution time are based on the average performance over each set of three problems. For this Phase I testing, we evaluated the following methods and variations:

- A1t: Algorithm A1 with tie-breaking rule.
- A1n: A1 without tie-breaking;
- A2t: Algorithm A2 with tie-breaking rule.
- A2n: A2 without tie-breaking;
- V*i*t: Algorithm V*i* with tie-breaking rule, for $i = 1, 2, 3, 4, 5$.
- V*i*n: V*i* without tie-breaking;
- DDT: DDT algorithm.

Before launching into the comparative testing, some preliminary testing was undertaken to generally try out the methods and to resolve certain open issues regarding methods A3, V4 and V5. Initially, we investigated parameter settings for method A3 with problems of 1000 variables. A3, due to poor performance, was eliminated from Phase I consideration as a stand-alone solution method. However, a version of A3 is used as a tie-breaking rule for the rest of algorithms. The two key issues to be resolved in applying A3 are the choice of the function $F(c)$ and the fraction, f, to be used in the voting procedure. We tested several implementations of A3 by changing α in $F(c) = c^{\alpha}$, and experimenting with the value of the voting fraction over the interval $[0, 1]$. Our results indicated that neither the solution quality nor solution time were significantly affected by different values for α or the voting fraction. For the purpose of serving as a tie-breaking rule in the other heuristics, we used $\alpha = 2$, and $f = 1$, which performed reasonably well in our initial testing.

Another preliminary issue concerns the choice between the "pairwise" and "most negative" strategies for use in methods V4 and V5. To resolve this issue we did experiments on problems of 1000 variables with no tie-breaking rule. For both V4 and V5 we found that the pairwise strategy worked far better than the most negative element strategy. As a result, all further testing was carried out utilizing the pairwise strategy only.

After handling the issues noted above, the comparative testing of the various methods was undertaken. Early testing on small problems (1500 variables or fewer) indicated that all the "V" methods, except V3n and V3t, were consistently dominated in terms of solution quality by the remaining methods. (Additional testing on other problems confirmed this dominance.). As a result, these methods were dropped from further testing and Phase I was continued using the six most promising new methods, A1t, A1n, A2t, A2n, V3t, V3n, along with the standard DDT method.

The results of the testing are summarized graphically in Figs. 2–9. All codes were written in

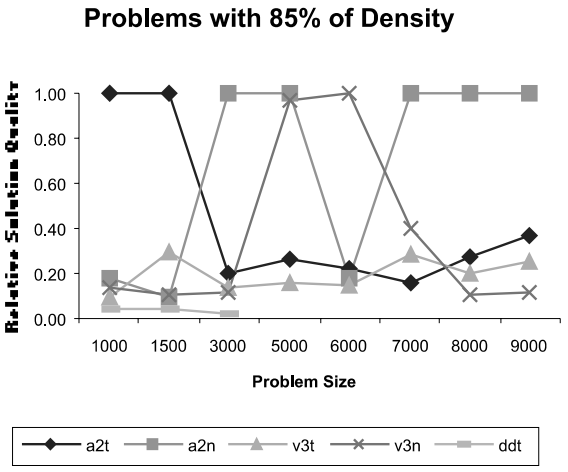

Fig. 2. Solution quality for 85% density problems.

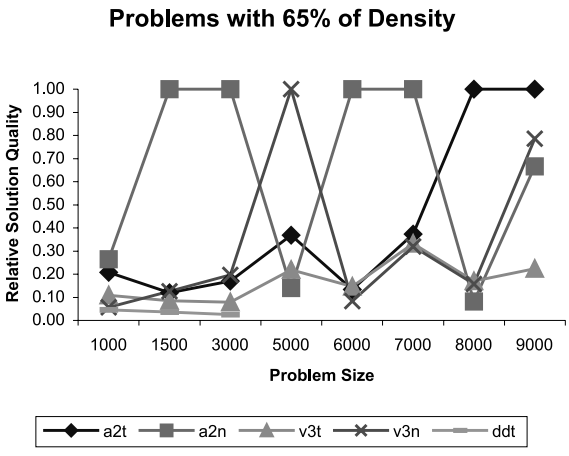

Fig. 4. Solution quality for 65% density problems.

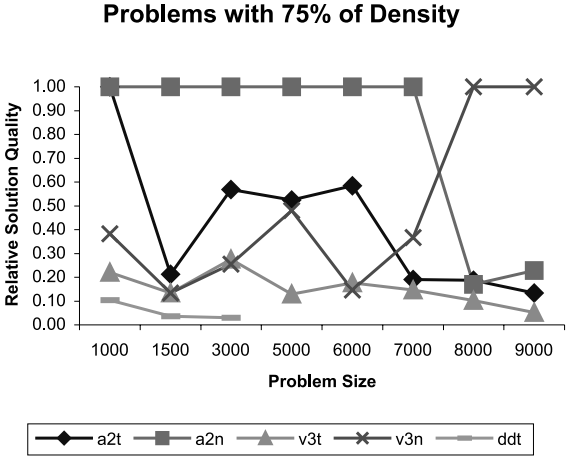

Fig. 3. Solution quality for 75% density problems.

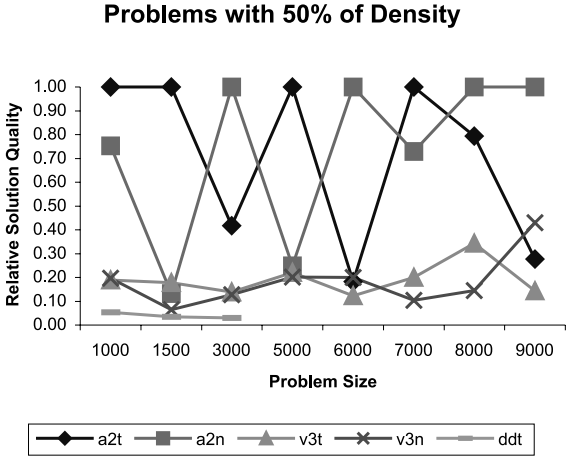

Fig. 5. Solution quality for 50% density problems.

F90 Fortran and run on a Cray C916. Although this is a 10 processor computer, no explicit parallel computing is used in our codes. The relative solution quality produced by the one-pass methods tested is presented in Figs. 2–8. We should mention that the relative difference in the quality of the algorithms for one problem is scaled according to the relative percentage obtained on the overall problems of the same density.

In each of the Figs. 2–8, the points plotted are averages over the set of three random instances for the size and density indicated. To facilitate making comparisons, the best result for each problem class is denoted as 1.0 and the performance of the other methods is plotted relative to this benchmark of 1.0. The results for methods A1t and A1n are not included in the graphs. Of the six new methods tested in Phase I, neither A1t nor A1n produced more than the 4th best results for any problem and most often they came in 5th and 6th. To avoid unnecessary clutter, we omitted the points for A1t and A1n from the graphs.

Computation times for the methods are indicated by the graph shown in Fig. 9, which gives

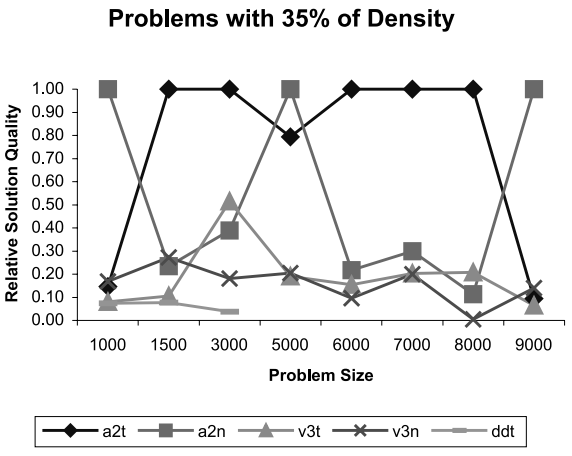

Fig. 6. Solution quality for 35% density problems.

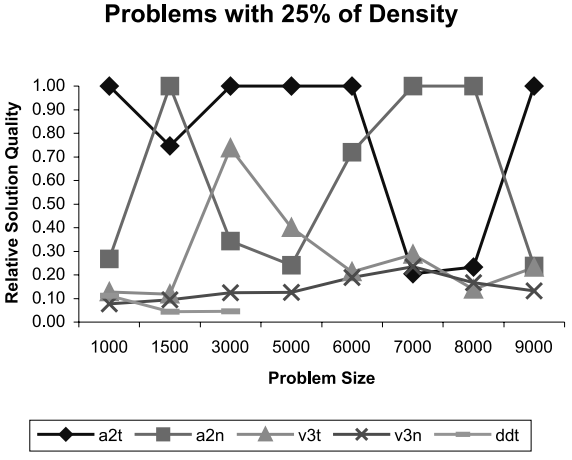

Fig. 7. Solution quality for 25% density problems.

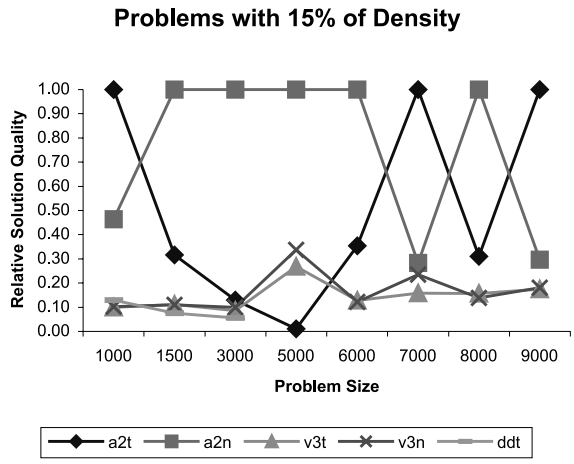

Fig. 8. Solution quality for 15% density problems.

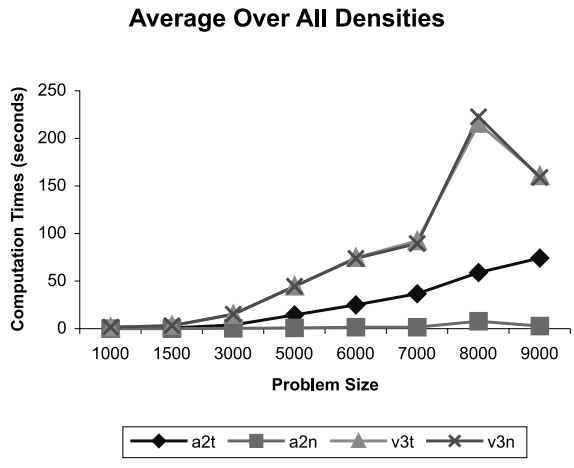

Fig. 9. Average CPU times over all densities.

plots of averages taken over all densities. Corresponding to the graphs of Figs. 2–8 showing relative solution quality, we constructed companion graphs showing average computational times for the methods on the respective problems. Without exception, these companion graphs exhibited the same pattern (shapes) shown in Fig. 9, differing from each other only in scale. Thus, in the interest of brevity, we chose to compress all computational times into a single figure representing the general shapes and to note that the scales for the individual charts are $[0, 140]$ for problems of densities of 65%, 75% and 85%, and $[0, 200]$, $[0, 250]$, $[0, 300]$, and $[0, 450]$, respectively, for problems of densities 50%, 35%, 25%, and 15%.

3.1.1. Analysis of Phase I results

The results in Figs. 2–8 indicate that no single method dominates in all cases. However, it is clear that methods A2t and A2n performed consistently and considerably better than the other methods along both dimensions of solution quality and computational time for the problems considered in

Phase I. The tie-breaking rule appears to improve the performance of method A2 for certain problems but has little positive effect on the solution quality of method V3. Perhaps the most noticeable result in the tables is the complimentary nature of methods A2t and A2n. Over the 56 "best results" reported in Figs. 2–8, 52 of them are given by either A2t or A2n. The remaining four best results were given by V3n.

Figs. 2–8 contain results for the DDT method for problem sizes up to 3000 variables only. Since the original DDT code was not available for our testing, we used our own DDT implementation which is well tested and proven to correctly execute the method in terms of solution quality. However, we suspect that a more efficient implementation can be made as our computation times for DDT were large even for small N. For the current version (which is based on signed graphs), the DDT running times were 82.3, 277.6, and 2205.4 seconds on average over all densities for problems of sizes 1000, 2000 and 3000, respectively (more than on order of magnitude slower than the next slowest method, V3). DDT was not included in the testing on problems larger than 3000 variables due to these long run times.

Quite aside from the solution times, we were surprised at the solution quality produced by DDT for the problems tested in Phase I where it generally lagged behind the other methods tested. Our previous experience with our version of DDT led us to expect that DDT would do much better. As we will see in the Phase II results that follow, there are many problems for which DDT shows extraordinary performance, often yielding results superior to the methods that outperformed it on the Phase I problems. Nonetheless, we take the results reported here in Phase I as strong support for our earlier conjecture that variations that fix variables one at a time, rather than undertaking (as in DDT) to make multiple assignments on a single step, can yield improved performance for many problems.

The relationship between computation time and problem size and density varies dramatically depending on the method. For method V3, computation time grows rapidly with problem size as shown in Fig. 9. However, computation time for a given problem size *decreases* by roughly a factor of one third as density goes from 15% to 85%. Method A2 behaves quite differently. Computation times for A2t increase more moderately than that of method V3 as problem size grows (see Fig. 9). For example, A2t and V3t solve 1000 variable problems in roughly 0.3 and 1.5 seconds, respectively, while taking 74 and 160 seconds, respectively, for 9000 variable problems (average values across all densities).

For a given problem size, computation times for A2t *increase* with density, going up by roughly a factor of 2 as density goes from 15% to 85%. Method A2n has the smallest computation times of all the methods. As shown in the Table 9, computation time increases very slightly with problem size. For example, A2n solves 1000 variable problems in roughly 0.04 seconds and 9000 variable problems in 2.7 seconds (averages over all densities). Moreover, for a given problem size, computation times are essentially constant as density goes from 15% to 85%. (Subsequent testing, not shown in the tables, disclosed that A2n generates solutions for 85% dense problems with 12,000 variables in less that 17 seconds.)

Computation times for A1t and A1n were slightly higher (a few percent) than those for A2t and A2n, respectively. Generally, they exhibited the same pattern with size and density as for A2t and A2n, respectively. As already noted, however, A1t and A1n lagged behind the other four new methods on the Phase I problems in terms of solutions quality.

Based on the full range of computational testing conducted in Phase I, we conclude that both methods A2 and V3 are effective for the problems considered here. Considering both solution quality and computation time, A2n gave the overall best performance, followed closely by A2t and V3n and V3t in that order.

3.2. Phase II. Comparing one-pass performance with best-known solutions

Phase II of our testing undertakes to determine how the methods compare in terms of solution quality on problems from the literature for which

known best solutions are available. For this purpose we took what appears to be the best of the methods from Phase I, A2n, A2t, V3n and V3t, and applied these four one-pass methods along with DDT to 98 problems from the open literature. DDT was included here due to the favorable performance experience we previously had with it on other problems and the fact that it is prominent in the literature.

Table 1 gives references for the problems used. These problems vary considerably in size, density, and in the characteristics of their Q matrices.

The results obtained from solving these test problems are shown in Tables 2–13. Each table is constructed in the following manner. The first column identifies the problem, followed by information on size, density and best-known solution. The last five columns give the results for DDT, A2n, A2t, V3n, and V3t, respectively, as a percentage of the best-known solution. For example, Table 2, row 1 shows that problem B1 has 20 variables, is 100% dense, and has a best-known solution value of 133. DDT gave a solution value that is 73.7% of 133 (or 98) and A2n gave a solution value that is 100% of the best-known result (133), while A2t, V3n and V3t gave results that were 78.9%, 10.5%, and 10.5%, respectively, of the best-known result.

3.2.1. Analysis of phase II results

Tables 2–13 indicate that the five one-pass methods tested in Phase II, overall, did well on the problems tested here. DDT in particular, which did not fare well in our Phase I testing, did extraordinarily well in Phase II. Nonetheless, as we saw in Phase I, no one method dominates in all cases.

Over the 98 problems solved in Phase II, the objective function values obtained by DDT averaged 91.6% of the best-known objective function values. For the 60 "Beasley" problems (Tables 9–13), DDT objective function values averaged 98.9% of the best-known values and matched the best-known values on nine of the problems. On the first 38 test problems (Tables 2–7), DDT results averaged 80% of the best-known values. Relative to the five heuristics tested, DDT gave the best result for 72 of the 98 problems considered.

For the entire problem set (98 problems), A2n objective function values averaged 82% of the best-known values, dropping slightly to 80% for the 60 Beasley problems. For the first 38 problems (Tables 2–7), A2n values averaged 85% of the best-known values, matching the best-known values 6 times. A2t, the A2 version with tie-breaking, gave an average result of 79.8% of the best-known solution overall, 80.1% for Beasley's problems, and

Table 1
Characteristics of test problems

Problem type	Source	Main diagonal	Off diagonal
B	Pardalos and Rodgers [22][a]	$[-63, 0]$	$[1, 100]$
F1	Glover et al. [9][b]	$[-75, +75]$	$[-50, +50]$
F2	Kochenberger et al. [16][b]	$[-99, 0]$	$[0, +50]$
G1	Glover et al. [9][b]	$[-75, +75]$	$[-50, +50]$
G2	Kochenberger et al. [16][b]	$[-99, 0]$	$[0, +50]$
MYCIEL	Michael Trick[c]	Negative	Non-negative
QUEEN	Michael Trick[c]	Negative	Non-negative
50	OR-Library[d]	$[-100, +100]$	$[-100, +100]$
100	OR-Library[d]	$[-100, +100]$	$[-100, +100]$
250	OR-Library[d]	$[-100, +100]$	$[-100, +100]$
500	OR-Library[d]	$[-100, +100]$	$[-100, +100]$
1000	OR-Library[d]	$[-100, +100]$	$[-100, +100]$
2500	OR-Library[d]	$[-100, +100]$	$[-100, +100]$

[a] Generator Q01SUBS available at http://mcs.anl.gov/home/otc/Server.
[b] Problems available at Hearin Center http://hces.bus.olemiss.edu.
[c] Graph coloring problems reformulated as xQx. Problems available at http://mat.gsia.cmu.edu/COLOR/instances.html.
[d] Problems available OR-Library http://mscmga.ms.ic.ac.uk/info.html.

Table 2
"B" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
B1	20	100	133	73.7	100.0	78.9	10.5	10.5
B2	30	100	121	95.0	75.2	86.8	5.0	5.0
B3	40	100	118	47.5	86.4	80.5	0.0	0.0
B4	50	100	129	66.7	78.3	78.3	0.0	0.0
B5	60	100	150	70.0	100.0	60.0	0.7	0.7
B6	70	100	146	43.2	77.4	72.6	2.7	2.7
B7	80	100	160	56.3	100.0	100.0	0.6	0.6
B8	90	100	145	61.4	80.7	80.7	0.0	0.0
B9	100	100	137	70.1	92.7	75.9	0.0	0.0
B10	125	100	154	65.6	78.6	78.6	0.0	0.0

Table 3
"F1" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
F1a	500	10	61,194	99.2	77.9	76.9	92.9	92.8
F1b	500	25	100,161	99.3	80.4	80.8	92.7	90.4
F1c	500	50	138,035	98.7	78.6	79.7	92.7	92.1
F1d	500	75	172,771	98.7	82.0	83.9	94.4	94.4
F1e	500	100	190,507	98.9	87.8	77.5	95.4	94.2

Table 4
"F2" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
F2a	500	10	3955	79.0	87.5	84.0	80.0	80.0
F2b	500	25	1998	81.3	92.4	92.4	73.4	89.8
F2c	500	50	1086	78.7	85.9	85.9	72.5	82.6
F2d	500	75	685	80.3	89.6	73.7	66.3	73.6
F2e	500	100	418	68.4	84.4	76.8	76.8	76.8

Table 5
"G1" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
G1a	1000	10	131,456	98.6	75.8	77.7	91.7	88.0
G1b	1000	30	192,565	96.4	74.2	67.0	79.0	84.4
G1c	1000	50	242,367	96.4	79.2	54.8	69.5	89.1
G1d	1000	70	253,590	96.0	68.0	60.9	75.8	86.8
G1e	1000	100	274,375	96.2	70.1	68.0	77.2	83.7

Table 6
"G2" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
G2a	1000	10	4929	81.5	92.8	85.8	84.2	77.4
G2b	1000	30	2050	73.8	85.5	81.9	77.1	77.1
G2c	1000	50	1241	66.6	80.3	88.8	77.2	83.9
G2d	1000	70	843	67.1	85.5	80.4	74.0	76.6
G2e	1000	100	452	64.8	88.1	86.9	78.5	73.9

Table 7
Graph coloring problems formulated as xQx

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
MYCIEL3	96	13.0	59	93.2	100.0	96.6	20.3	100.0
MYCIEL4	240	7.3	120	95.8	100.0	97.5	11.7	100.0
MYCIEL5	480	4.5	239	92.1	100.0	98.7	5.9	100.0
QUEEN5	125	14.2	125	76.0	80.0	60.0	4.0	76.0
QUEEN6	252	9.2	180	83.3	83.3	72.2	2.8	83.3
QUEEN7	343	7.7	245	73.5	79.6	67.3	2.0	69.4
QUEEN8	640	5.1	320	79.7	93.8	81.3	1.6	84.4
QUEEN9	810	4.5	405	76.5	84.0	75.3	1.2	75.3

Table 8
"50" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
50.01	50	10	2098	90.8	74.9	79.1	32.5	32.5
50.02	50	10	3702	100.0	60.2	60.2	0.0	0.0
50.03	50	10	4626	100.0	85.2	85.2	76.0	76.0
50.04	50	10	3544	97.9	86.9	86.9	81.3	81.3
50.05	50	10	4012	99.5	76.7	76.7	96.8	96.8
50.06	50	10	3693	100.0	71.1	71.1	91.3	91.3
50.07	50	10	4520	100.0	75.5	75.5	94.0	94.0
50.08	50	10	4216	100.0	76.5	76.5	86.9	79.5
50.09	50	10	3780	99.2	54.6	54.6	75.6	75.6
50.1	50	10	3507	99.7	78.0	78.0	92.7	92.7

79.34% on the first 38 problems. Nowhere in the 98 problems did A2t match the best-known result. Compared to the other four heuristics, A2n gave the best result on 26 of the 98 problems tested, while A2t gave the best result for only one problem instance.

V3n, over the entire problem set, gave an average performance result of 71.6% of the best-known results. This average was brought down considerably by the poor performance on the problems of Tables 2 and 7. Performance on the first 38 problems, due the problems of Tables 2 and 7, was only 44.5% of the best-known results. For the 60 Beasley problems, V3n gave an average 88.8% of best-known values. Nowhere in the entire problem set did V3n match the best-known result.

Table 9
"100" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
100.01	100	10	7970	97.7	78.8	78.4	85.3	85.3
100.02	100	10	11,036	100.0	81.5	81.5	93.8	93.8
100.03	100	10	12,723	99.6	81.2	90.9	99.5	99.5
100.04	100	10	10,368	100.0	88.5	86.4	92.4	92.4
100.05	100	10	9083	100.0	76.8	76.8	90.9	90.9
100.06	100	10	10,210	97.3	77.8	77.8	83.6	83.6
100.07	100	10	10,125	99.8	80.0	80.0	85.9	66.2
100.08	100	10	11,435	100.0	83.2	83.2	93.0	93.0
100.09	100	10	11,455	99.4	70.7	70.7	82.5	82.6
100.1	100	10	12,656	98.8	77.5	77.5	96.1	96.8

Table 10
"250" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
250.01	250	10	45,607	99.3	90.2	90.5	93.0	93.1
250.02	250	10	44,810	94.2	82.5	80.5	92.8	88.3
250.03	250	10	49,037	99.9	86.4	86.4	97.2	97.2
250.04	250	10	41,274	99.5	90.0	89.9	89.2	94.9
250.05	250	10	47,961	99.7	83.4	84.2	92.7	96.3
250.06	250	10	41,014	98.8	78.5	78.5	89.8	88.7
250.07	250	10	46,757	99.3	71.3	71.3	92.4	92.4
250.08	250	10	35,726	98.1	81.6	81.6	92.7	92.9
250.09	250	10	48,916	99.0	80.3	79.8	97.5	97.5
250.1	250	10	40,442	98.6	77.4	78.5	94.5	93.4

Table 11
"500" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
500.01	500	10	116,586	98.5	81.4	77.2	86.7	91.1
500.02	500	10	128,223	99.4	74.8	79.9	94.3	91.2
500.03	500	10	130,812	99.4	83.6	78.3	89.7	87.3
500.04	500	10	130,097	99.4	77.2	76.3	93.3	93.4
500.05	500	10	125,487	99.0	79.6	77.3	92.7	94.0
500.06	500	10	121,719	99.2	83.3	83.3	95.1	12.6
500.07	500	10	122,201	98.2	79.8	79.2	93.6	93.6
500.08	500	10	123,559	99.4	80.4	80.0	91.4	90.8
500.09	500	10	120,798	99.0	79.0	82.0	93.7	93.7
500.1	500	10	130,619	99.3	84.9	84.7	93.6	93.4

V3t, the V3 version with tie-breaking, gave an average result of 78% overall, 87.5% for the Beasley problems, and 63% on the first 38 problems. V3t matched the best-known result on three problem instances. Overall, the V3 methods exhibited much more volatility in performance than

Table 12
"1000" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
1000.01	1000	10	371,438	99.2	79.4	80.4	91.0	91.8
1000.02	1000	10	354,932	99.0	84.0	84.0	93.1	93.1
1000.03	1000	10	371,226	99.5	85.7	85.5	95.3	95.1
1000.04	1000	10	370,560	99.0	83.6	83.5	94.6	94.4
1000.05	1000	10	352,736	98.1	82.9	81.9	93.0	92.9
1000.06	1000	10	359,452	99.2	82.9	83.4	93.9	95.7
1000.07	1000	10	370,999	99.3	78.6	78.3	65.2	91.8
1000.08	1000	10	351,836	98.7	81.8	84.3	93.8	93.8
1000.09	1000	10	348,732	99.0	83.1	80.9	91.9	91.9
1000.1	1000	10	351,415	99.1	83.4	79.9	93.1	93.1

Table 13
"2500" problems

Problem	N	Density (%)	Best known	Percent of best known				
				DDT	A2n	A2t	V3n	V3t
2500.01	2500	10	1,515,011	99.1	81.0	78.3	93.7	93.1
2500.02	2500	10	1,468,850	99.1	81.5	81.5	94.7	94.3
2500.03	2500	10	1,413,083	99.1	79.5	83.9	93.9	94.6
2500.04	2500	10	1,506,943	99.4	82.3	82.6	92.6	94.2
2500.05	2500	10	1,491,796	99.1	82.4	82.1	94.0	94.1
2500.06	2500	10	1,468,427	99.4	79.5	81.0	94.3	93.5
2500.07	2500	10	1,478,654	99.1	82.3	81.1	94.7	93.2
2500.08	2500	10	1,484,199	99.3	82.6	83.0	94.4	93.0
2500.09	2500	10	1,482,306	99.1	80.4	79.7	92.5	92.1
2500.1	2500	10	1,482,354	99.3	83.8	83.1	94.8	93.8

either DDT or the A2 methods. Moreover, while the V3 methods were competitive on many problems, they never gave a result better than the other three methods (DDT, A2n or A2t) for any of the 98 test problems. V3n was not able to match the best performance of the other methods on any of the 98 problems and V3t achieved this for only four of the 98 problems (all from Table 7]. Note that method A2n also had these best-known results for these same four problems. Over the entire problem set, neither of the V3 methods produced an objective function result better than that of the other three methods for any particular problem.

Generally speaking, the best solution obtained for each of the 98 problems was obtained by either DDT or A2n. The only exception to this was problem G2c (Table 6), where the best result was given by A2t. Overall, DDT outperformed A2n on the Beasley problems and the problems of Tables 3 and 5 while A2n outperformed DDT on the problems of Tables 2, 4, 6, and 7. It is interesting to note that the better of the two results for each problem of Phase II (i.e., from either DDT or A2n) was 95.91% of the best-known solution over the entire 98 problems. Seldom (only once) did they tie for best result, instead exhibiting a complementarity having one method dominate for certain problems and the other method for other problems. This complementary behavior is highlighted with respect to best-known solutions with DDT matching best-known solutions on nine of the Beasley problems and A2n matching best-known solutions on six of the other problems.

DDT's strong performance on the Beasley problems is consistent with our expectation (based on earlier testing prior to this study) that the

method performs very well on sparse problems and on problems with unrestricted q_{ij} values (characteristics exhibited by Beasley's problems). As shown in Tables 2, 4, 6 and 7, A2n seems to do well on problems with negative main diagonal elements and nonnegative off diagonal elements, a structure that shows up in many applications. Nonetheless, there are many exceptions that suggest the relationship between structure and method performance is more complicated. For example, all 168 problems of Phase I were generated with unrestricted q_{ij} values and yet DDT, even on the sparse instances, did much worse than A2n (and other methods) on these problems. While the computational results presented in this paper firmly establish the effectiveness of one-pass methods as a class of heuristics, there is much left to learn about which method is best for a given (new) problem instance.

4. Summary and conclusion

Our study introduces and tests several one-pass heuristics for large-scale instances of the unconstrained binary quadratic programming problem (UQP). Rigorous computational experiments were conducted on problems with up to 9000 variables, which translate into linear mixed integer programming problems containing over 40,000,000 variables. This is the first study to report extensive experience with one-pass methods on such a large and diverse set of problems, whose sizes range significantly beyond those previously examined in the literature.

Our computational work discloses that several of the one-pass methods are very effective for solving UQP. While no single method dominated in every instance, five methods, A2n, A2t, V3n, V3t, and DDT stand out as being particularly effective for certain problem instances. Considering both solution quality and solution time, method A2 (A2n in particular) gave the best overall performance in our study. DDT, while taking considerably longer than A2n, produced very high quality solutions on problems up to 2500 variables and clearly demonstrated its effectiveness, particularly on sparse problems.

Our on-going research centers around new applications that involve UQP problems even larger than those considered here. To accommodate such applications we are testing new data structures and procedural enhancements designed to further improve computational times. The success reported in this paper on the rapid execution and high quality solution characteristics of one-pass methods motivates us to examine "master methods" that employ several methods in concert. Specifically, we are testing the effective bundling of the best one-pass heuristics into a suite of solvers that operate in a multi-pass fashion to rapidly produce several candidate solutions to a given problem, selecting the best of these as the final outcome. The various solutions generated can also serve as starting points for more advanced methods. We hope to report on these and related issues in future papers.

Acknowledgements

This research was supported in part by ONR grants # N000140010598 and N000140010769.

References

[1] B. Alidaee, G. Kochenberger, A. Ahmadian, 0–1 Quadratic programming approach for the optimal solution of two scheduling problems, International Journal of Systems Science 25 (1994) 401–408.
[2] T.M. Alkhamis, M. Hasan, M.A. Ahmed, Simulated annealing for the unconstrained binary quadratic pseudo-boolean function, European Journal of Operational Research 108 (1998) 641–652.
[3] M. Amini, B. Alidaee, G. Kochenberger, A scatter search approach to unconstrained quadratic binary programs, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Methods in Optimization, McGraw-Hill, New york, 1999 (to appear).
[4] J.E. Beasley, Heuristic algorithms for the unconstrained binary quadratic programming problem, Working Paper, Imperial College, 1999.
[5] E. Boros, P. Hammer, X. Sun, The DDT method for quadratic 0–1 minimization, RUTCOR Research Center, RRR 39-89, 1989.
[6] P. Chartaire, A. Sutter, A decomposition method for quadratic 0–1 programming, Management Science 41 (4) (1994) 704–712.

[7] G. Gallo, P. Hammer, B. Simeone, Quadratic knapsack problems, Mathematical Programming 12 (1980) 132–149.
[8] F. Glover, M. Amini, G. Kochenberger, B. Alidaee, A new evolutionary metaheuristic for the unconstrained binary quadratic programming: A case study of the scatter search, School of Business, University of Colorado, Boulder, September 1999.
[9] F. Glover, G. Kochenberger, B. Alidaee, M.M. Amini, Tabu with search critical event memory: An enhanced application for binary quadratic programs, in: S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1999.
[10] F. Glover, G. Kochenberger, B. Alidaee, M. Amini, Unconstrained quadratic binary program approach to quadratic Knapsack problems, Working paper, Hearin Center for Enterprise Science, University of Mississippi, 1999.
[11] F. Glover, G. Kochenberger, B. Alidaee, Adaptive memory tabu search for binary quadratic programs, Management Science 44 (3) (1998) 336–345.
[12] P. Hammer, E. Boros, X. Sun, On quadratic unconstrained binary optimization, INFORMS National Meeting, Seattle, October, 1998.
[13] P. Hammer, S. Rudeanu, Boolean Methods in Operations Research, Springer, New York, 1968.
[14] F. Harary, On the notion of balanced of a signed graph, Michigan Mathematical Journal 2 (1953/54) 143–146.
[15] K. Katayama, M. Tani, H. Narihisa, Solving large binary quadratic programming problems by an effective genetic local search algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'00), Morgan Kaufmann, 2000 (to appear).
[16] G. Kochenberger, B. Alidaee, M. Amini, Applications of the unconstrained quadratic binary program, Working Paper, University of Colorado, 1998.
[17] J. Krarup, A. Pruzan, Computer aided layout design, Mathematical Programming Study 9 (1978) 75–94.
[18] D.J. Laughunn, Quadratic binary programming, Operations Research 14 (1970) 454–461.
[19] A. Lodi, K. Allemand, T.M. Liebling, An evolutionary heuristic for quadratic 0–1 programming, Technical Report OR-97-12, D.E.I.S., University of Bologna, 1997.
[20] R.D. McBride, J.S. Yormack, An implicit enumeration algorithm for quadratic integer programming, Management Science 26 (1980) 282–296.
[21] P. Merz, B. Freisleben, Genetic algorithms for binary quadratic programming, in: Proceedings of the 1999 International Genetic and Evolutionary Computation Conferecne (GECCO'99), Morgan Kaufmann, Los Altos, CA, 1999, pp. 417–424.
[22] P. Pardalos, G.P. Rodgers, Computational aspects of a branch and bound algorithm for quadratic 0–1 programming, Computing 45 (1990) 131–144.
[23] P. Pardalos, G.P. Rodgers, A branch and bound algorithm for maximum clique problem, Computer & OR 19 (1992) 363–375.
[24] P. Pardalos, J. Xue, The maximum clique problem, The Journal of Global Optimization 4 (1994) 301–328.
[25] A.T. Phillips, J.B. Rosen, A quadratic assignment formulation of the molecular conformation problem, Journal of Global Optimization 4 (1994) 229–241.
[26] A.C. Willaims, Quadratic 0–1 programming using the roof duality with computational results, Rutcor Research Report 8-85, Rutgers University, New Brunswick, NJ, 1985.
[27] C. Witsgall, Mathematical methods of site selection for electronic system (EMS), NBS Internal Report, 1975.