World Academy of Science, Engineering and Technology
International Journal of Industrial and Manufacturing Engineering
Vol:2, No:1, 2008

Perturbation Based Search Method for Solving Unconstrained Binary Quadratic Programming Problem

Muthu Solayappan, Kien Ming Ng, and Kim Leng Poh

Abstract—This paper presents a perturbation based search method to solve the unconstrained binary quadratic programming problem. The proposed algorithm was tested with some of the standard test problems and the results are reported for 10 instances of 50, 100, 250, & 500 variable problems. A comparison of the performance of the proposed algorithm with other heuristics and optimization software is made. Based on the results, it was found that the proposed algorithm is computationally inexpensive and the solutions obtained match the best known solutions for smaller sized problems. For larger instances, the algorithm is capable of finding a solution within 0.11% of the best known solution. Apart from being used as a stand-alone method, this algorithm could also be incorporated with other heuristics to find better solutions.

Keywords—unconstrained binary quadratic programming, perturbation, interior point methods

I. Introduction

QUADRATIC Programming is a special type of mathematical optimization problem, which involves minimization of a quadratic function subject to linear constraints. Unconstrained binary quadratic programming problem (UBQP) refers to minimizing a quadratic function subject to variables being 0 or 1. A general formulation of the UBQP problem is given below:

$$\begin{aligned} \min\ & f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} \\ \text{s.t. } & \mathbf{x} \in \{0,1\}^n \end{aligned} \tag{1}$$

where $\mathbf{A}$ is a symmetric $n \times n$ matrix. It is an NP-hard problem and has a multitude of applications ranging from the problem of ranking a sports team [1] to that of determining the native conformation of molecules [2]. Most of the quadratic integer programming problems resulting from different problem scenarios could be formulated as a standard unconstrained binary quadratic programming problem, and hence developing solution techniques for problems of this type serves a much wider purpose.

In spite of the rich literature available on solving the problems of type (1), most of the exact methods proposed are based on the branch and bound technique varying in the development of cuts and bounds, forcing rules and preprocessing techniques. Solution methods thus developed rely on the quality of bounds or the efficiency of cuts generated to constrain the feasible region. Moreover, only medium sized problems were attempted. On the other hand, heuristics such as simulated annealing, tabu search and genetic algorithm, coupled with hybrid approaches were more often used. Perturbation methods, though commonly used to solve problems that are not amenable to exact methods, have been uncommon in the area of unconstrained binary quadratic programming problems. As perturbation methods may have certain strengths that apply to our problem of interest, we have developed a solution technique using such methods coupled with a particular search direction to solve problems of type (1). In particular, this technique involves solving the relaxed version of problem (1) by suitably perturbing the matrix $\mathbf{A}$. Such a perturbation gives us a chance to explore the neighborhood of the current iterate to look for improved solutions, with the possibility of avoiding the situation of being trapped at local optimal solutions.

The rest of the paper is organized as follows: Section II presents a brief literature survey of methods used to solve problems of type (1), whereas section III introduces the problem statement together with some background on the perturbation approach and the direction of search. The perturbation based search method that is being proposed is then explained in detail in section IV. Section V provides a brief note on the parameter initializations. Numerical results and the performance of the algorithm are discussed in section VI. Section VII provides concluding remarks and areas of future research.

Muthu Solayappan (muthu@nus.edu.sg), Kien Ming Ng (isenkm@nus.edu.sg) and Kim Leng Poh (isepohkl@nus.edu.sg) are with the Department of Industrial and Systems Engineering, National University of Singapore.

II. Related Work

UBQP has been in existence for a long period of time and various solution techniques, both exact methods and heuristics, have been proposed. The branch and bound method has been used to develop exact solution techniques for solving UBQP [3]–[5], and [6] develops an approximate algorithm using bounding techniques to solve the UBQP problem. Semidefinite relaxation of UBQP coupled with the addition of cuts to the feasible region [7] proves to be an effective method when solved by the ellipsoidal algorithm for moderately sized problems.

Variants of tabu search, simulated annealing and genetic algorithm have been extensively used to solve UBQP. Tabu search involving strategic oscillation between adding and dropping variables progressively produced some excellent results [8]. A similar two-phase approach has been used to solve the UBQP reformulation of uncapacitated task allocation problem

[9] and the vertex coloring problem [10]. Simulated annealing based on a local search with a simple cooling schedule and multiple annealing process starting from different temperatures, though successfully applied to the UBQP, is limited by its running time [11]. Solution techniques inspired from the ideas of genetic algorithm have been used to solve quadratic assignment problems [12], [13], which are closely related to the UBQP.

III. Problem Statement

The problem that we are trying to solve is similar to the standard formulation of the unconstrained binary quadratic programming problem except that the objective is to maximize the quadratic function. In order to solve problem (1), its relaxed version is solved with the hope that the solution would converge to integral values by an appropriate choice of perturbation strategies. The relaxed version of the problem of interest is stated below:

$$\begin{aligned} \max\ & f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x} \\ \text{s.t. } & \mathbf{0} \le \mathbf{x} \le \mathbf{e} \end{aligned} \tag{2}$$

where $\mathbf{e}$ is an n-dimensional vector of 1's, $\mathbf{A}$ is an $n \times n$ symmetric matrix and $\mathbf{x}$ is an n-dimensional vector.

A. Perturbation

In order to explore the neighborhood of the current iterate and and its potential to escape some local minima, we looked at different strategies of perturbation to be employed, while searching in a particular descent direction. We perturb the matrix $\mathbf{A}$ by a small quantity and see if this change could lead to a better solution. The perturbation of matrix $\mathbf{A}$ alters the landscape of the original objective function, and masks the original local minima points, thereby enabling the search process to be more efficient. This helps in handling the local minima points, which otherwise would have become more difficult for the solution process. However, the perturbed problem would have its own local minima points, which is countered by iteratively updating the perturbation parameter based on the values of the $\mathbf{x}$-vector. The perturbed version of the problem formulation is given below:

$$\begin{aligned} \min\ & f_p(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} \\ \text{s.t. } & \mathbf{0} \le \mathbf{x} \le \mathbf{e} \end{aligned} \tag{3}$$

where $\mathbf{Q} = -\mathbf{A} + \mathbf{P}$ and $\mathbf{P}$ is an $n \times n$ matrix used for perturbing the matrix $\mathbf{A}$. Entries of the matrix $\mathbf{P}$ are randomly generated such that $0 \le P_{ij} \le 1$. The solution method that is being proposed is applied to this perturbed version with a view to solve the problem formulation in (1).

B. Direction of search

The objective function in (3) cannot be classified as convex or nonconvex due to the varying nature of matrix $\mathbf{Q}$. Only when $\mathbf{Q}$ is positive semidefinite, the objective function is convex and search directions could be derived from the first principles. This cannot always be the case as the entries of the matrix $\mathbf{Q}$ are problem specific. In order to get out of this conundrum, we use search directions that were derived for problems of similar nature in [14]. This particular reference deals with solving problems of the form $\frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{c}^T\mathbf{x},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u}$. They convert the problem to a convex optimization problem by adding a barrier function to the objective and hence obtain the search direction from first principles. We intend to use the same search direction which is also proved to be a direction of descent.

IV. Proposed Solution Method

The original unconstrained binary quadratic programming problem has been re-formulated as a continuous nonlinear programming problem with a perturbed objective function as shown in formulation (3). As the problem is unconstrained, any of the search techniques which takes into account the bounds on the variables can be used to solve the problem. However, such methods would not always guarantee to give a global optimum or near global optimum solution. In order to avoid getting trapped at poor solutions or exploring the feasible region along poor directions, we propose a solution method which may give a solution in the ϵ-neighborhood of the global optimum solution.

For a randomly generated initial solution, $\mathbf{0} < \mathbf{x} < \mathbf{e}$, the perturbed problem could be solved by moving in the direction of descent obtained from [14]. The descent direction that is being used is given below:

$$\begin{aligned} \text{descent direction, } & \mathbf{dir}(\mathbf{x}) = \mathbf{d}(\mathbf{x}) - \mathbf{x} \\ \text{where } & d_i(\mathbf{x}) = \frac{u_i + l_i\gamma_i(\mathbf{x})}{1+\gamma_i(\mathbf{x})} \\ & \gamma_i(\mathbf{x}) = \exp\left(\frac{1}{\beta}\frac{\partial f_\beta(\mathbf{x})}{\partial x_i}\right) \end{aligned} \tag{4}$$

The function f_β is a type of barrier function that has been well-experimented in [14] and as such, we will adapt it to solve the problem on hand. Based on the above-mentioned direction, coupled with perturbation of matrix $\mathbf{A}$, the solution method that is being proposed is shown in Algorithm 1. The barrier parameter, β, is an arbitrarily large value which is reduced at every iteration by a factor $\gamma \in (0, 1)$ when the current iterate is in the $\epsilon-$neighborhood of the stationary point. Generally the value of γ is set to a value close to 1, so that any drastic changes that might occur otherwise could be avoided. The values l_i and u_i in (4) represent the lower and upper bounds on the variable x_i, respectively. The method terminates when both the descent direction and barrier parameter are less than the given tolerance levels, ϵ and ϵ_β, respectively.

A search method is included in the algorithm to calculate the step length α for the calculated descent direction $\mathbf{dir}(\mathbf{x^k})$. This is often solved as a subproblem and could be stated as shown in (5):

$$\begin{aligned} \min\ & \varphi(\alpha) = f_p(\mathbf{x^k} + \alpha\mathbf{dir}(\mathbf{x^k})) \\ \text{s.t. } & 0 \le \alpha \le 1 \end{aligned} \tag{5}$$

This is a one-dimensional problem in the variable α and many methods are available to solve problems of this type. However, we prefer a method that does not require derivative information to reduce the overall computational time. Hence

Algorithm 1 Perturbation Based Search Method

```
Let $\beta_0$ = initial barrier parameter
    $\epsilon$ = tolerance for the function
    $\epsilon_\beta$ = tolerance for barrier parameter
    $\gamma$ = reduction factor
    $n$ = number of variables
    $\mathbf{P}$ = $n \times n$ matrix used for perturbing $\mathbf{A}$
    $k$ = iteration counter
    $\alpha$ = step length
    $\mathbf{x_0}$ = feasible starting vector

Set $flag$ = 0
    $k$ = 0
    $\beta$ = $\beta_0$
    $\gamma$ = any value in $(0, 1)$, preferably close to 1
while $flag = 0$
    Compute $\mathbf{Q} = -\mathbf{A} + \mathbf{P}$
    for $i = 1, \ldots, n$
            Compute the descent direction, $dir_i(\mathbf{x^k})$
    end for
    if $||\mathbf{dir}(\mathbf{x^k})|| \leq \epsilon$
            if $\beta > \epsilon_\beta$
                $\beta = \gamma\beta$
            else
                $flag = 1$
            end if
    else
            $\alpha$ = search_method()
            $x^{k+1} = x^k + \alpha dir(x^k)$
            $\mathbf{P}$ = perturbation_method()
            $k = k + 1$
    end if
end while
```

we have chosen the golden section method for minimizing the function $\varphi(\alpha)$ over the interval $[0, 1]$. The standard framework of the method has been used and for details of implementation, the reader is referred to [15]. Matrix $\mathbf{A}$ is perturbed at every iteration of Algorithm 1 by a separate procedure, $perturbation\_method()$, and the way it is implemented is detailed in Algorithm 2. For every vector $\mathbf{x}$ generated during the execution of the algorithm, the method assigns a small value δ to the i^{th} row and i^{th} column of the matrix $\mathbf{P}$ if the i^{th} component of vector $\mathbf{x}$ is neither 1 nor 0. The matrix $\mathbf{P}$ is then used to perturb the matrix $\mathbf{A}$ by using an additional operator. This would alter the matrix $\mathbf{A}$ thereby affecting the magnitude of the descent direction that is being generated. This would in turn help the algorithm to explore the neighborhood of the current iterate to see if better solutions could be found. At the same time, poor solutions could also be identified and discarded.

Algorithm 2 perturbation_method()

```
Let $\delta$ = perturbation parameter such that $0 < \delta < 1$
    $n$ = number of variables
    for $i = 1, \ldots, n$
      if $NOT(x(i) = 1\ OR\ x(i) = 0)$
          $i^{th}$ row and $i^{th}$ column of $\mathbf{P}$ is assigned $\delta$
      else
          $i^{th}$ row and $i^{th}$ column of $\mathbf{P}$ is assigned 0
      end if
    end for
```

V. Parameter Initialization

In order to start the proposed solution method, an initial interior feasible point must be provided. For our procedure, we randomly generate the starting vector $\mathbf{x}$ such that every component of $\mathbf{x}$ belongs to the set, $(0, 1)^n$. Since interior point methods such as the proposed method depend on the quality of the initial solution, we generate altogether 10 different starting vectors and the algorithm is run for each of these vectors. By doing so, it gives us an opportunity to select the best of 10 solutions provided by the algorithm.

The barrier parameter β is initially assigned an arbitrarily large value, say 1000, and is reduced at every iteration by a factor $\gamma \in (0, 1)$, preferably close to 1. For the golden section search method, the reduction ratio α is set at a constant value of $\frac{\sqrt{5}-1}{2}$ and the tolerance for the interval of uncertainty, l, is set at 0.01.

Though the magnitude of the objective function value is important, equally essential is the components of the solution vector being binary. Solving the relaxed version of the problem does not help to resolve this issue. However, this issue can be tackled by the perturbation approach we are proposing. As can be seen from Algorithm 2, a small value of δ is being added to every component of the solution vector that is not binary. This may serve as a penalty being added to the objective function for every component of the $\mathbf{x}$-vector that is not binary. As the objective function is to be minimized, the algorithm avoids the addition of the penalty δ by forcing the variables to either 0 or 1. Hence we almost always find the values of $\mathbf{x}$ to be binding. However, the value of δ is set to a relatively small value, ranging from 0.0001 to 0.01. This is because a large value of δ might alter the matrix $\mathbf{A}$ considerably, resulting in flawed search directions which would cause the algorithm to produce poor solutions. Consequently, in certain problem instances, we do have solution vectors with certain components being non-integral. In such cases, instead of simply rounding off the solution, different permutations of 0's and 1's are tried and the best solution is reported.

VI. Performance Evaluation

A. Numerical Results

In order to evaluate the performance of the algorithm, we conducted several experiments on 40 different problem instances contained in the OR-Library [16]. It presents a variety of test problems differing in the number of binary variables that are required to be solved. We tested our algorithm on problems of size n = 50, 100, 250 and 500 variables. The

TABLE I
COMPARISON OF OBJECTIVE FUNCTION VALUES.

Problem	Best Known Solution Reported by				Perturbation	%
Instance	TS-B	SA-B	GLS	SA-KN	Method	Difference
beas50-1	2098	2098	2098	2098	**2098**	0
beas50-2	3702	3702	3702	3702	**3702**	0
beas50-3	4626	4626	4626	4626	**4626**	0
beas50-4	3544	3544	3544	3544	**3544**	0
beas50-5	4012	4012	4012	4012	**4012**	0
beas50-6	3693	3693	3693	3693	**3693**	0
beas50-7	4520	4520	4520	4520	**4520**	0
beas50-8	4216	4216	4216	4216	**4216**	0
beas50-9	3780	3780	3780	3780	**3780**	0
beas50-10	3507	3507	3507	3507	**3507**	0
beas100-1	7970	7942	7970	**7970**	7904	0.83
beas100-2	11036	11036	11036	11036	**11036**	0
beas100-3	12723	12723	12723	12723	**12723**	0
beas100-4	10368	10368	10368	10368	**10368**	0
beas100-5	9083	9083	9083	9083	**9083**	0
beas100-6	10210	10210	10210	**10210**	10122	0.86
beas100-7	10125	10125	10125	**10125**	10098	0.27
beas100-8	11435	11435	11435	11435	**11435**	0
beas100-9	11455	11455	11455	11455	**11455**	0
beas100-10	12565	12565	12565	**12565**	12547	0.14
beas250-1	45607	45607	45607	**45607**	45579	0.06
beas250-2	44810	44810	44810	**44810**	44502	0.69
beas250-3	49037	49037	49037	**49037**	49019	0.04
beas250-4	41274	41274	41274	**41274**	41236	0.09
beas250-5	47961	47961	47961	**47961**	47948	0.03
beas250-6	41014	41014	41014	**41014**	40996	0.04
beas250-7	46757	46757	46757	46757	**46757**	0
beas250-8	35726	35726	35726	**35726**	35666	0.17
beas250-9	48916	48916	48916	**48916**	48733	0.37
beas250-10	40442	40442	40442	40442	**40442**	0
beas500-1	116586	116586	116586	**116586**	116452	0.11
beas500-2	128223	128204	128339	**128339**	128255	0.07
beas500-3	130812	130812	130812	130812	**130812**	0
beas500-4	130097	130077	130097	**130097**	130045	0.04
beas500-5	125487	125315	125487	**125487**	125397	0.07
beas500-6	121719	121719	121772	**121772**	121118	0.54
beas500-7	122201	122201	122201	**122201**	122159	0.03
beas500-8	123559	123469	123559	**123559**	123421	0.11
beas500-9	120798	120798	120798	**120798**	120616	0.15
beas500-10	130619	130619	130619	**130619**	130608	0.01

TS-B - Tabu Search as in [17]
SA-B - Simulated Annealing as in [17]
GLS - Genetic Local Search as presented in [18]
SA-KN - Simulated Annealing as presented in [19]

objective function to be maximized is of the form $\mathbf{x}^T\mathbf{Q}\mathbf{x}$, where $\mathbf{Q}$ is an $n \times n$ matrix with integer entries and density of 0.1.

Each and every problem type has 10 different instances and for each of these instances, the algorithm is run from 10 different starting vectors. Thus, a total of 400 runs of the algorithm were executed and the best solution for each problem instance is reported in Table I. Columns 2 to 5 of the table provide the best known solution value reported by [17]–[19]. The column titled, Perturbation Method, presents the solution that has been found using the proposed algorithm. The next column gives the percentage deviation by which our solution differs from the best known solution, calculated using $100(1 - \frac{f_{PM}}{f_{best}})$.

The algorithm was implemented in Matlab version 7.2. All the experimental runs were performed on the same Pentium IV PC running at 3.06GHz and 2GB of RAM. The algorithm was terminated when there is no improvement in the objective function value for a predefined number of iterations. Ten sets of initial solutions were randomly generated and the same sets were used for all the 10 instances of a particular problem type.

B. Comparison with other heuristics

In order to assess the performance of the proposed method, the numerical solutions obtained are compared with heuristic methods that have been developed to solve similar problems. Table I shows the best objective function value found by different heuristics and the best one among them is highlighted. The genetic local search proposed in [18] and the simulated annealing along with reannealing proposed in [19] matches the solution reported by [17] and in some cases even finds better solutions. Our method outperforms the simulated annealing method proposed in [17] for some instances. The solution proposed by our method is in the neighborhood of the best reported solution with an average deviation of less than 0.11%, even for the larger problem instances.

The above-mentioned heuristics, as is the case generally, build on a randomly generated solution through a preconditioning phase and then after finding a good solution, there is normally an improvement phase where a search for further improvement in solution quality is performed. The proposed perturbation method, however, lacks such complications and is comparatively easy to implement. Moreover, the time taken to obtain the final solution is significantly lesser when compared to the other methods. Table II compares the time taken (seconds) by different algorithms to that of the proposed method. It should be noted that the running times reported for [18] and [19] are for one phase only. Figure 1 compares the running time of our method to that of [17]. It is only natural to find that the running time increases with problem size. Direct empirical comparison of running time is not possible

TABLE II
COMPARISON OF RUNNING TIME WITH OTHER HEURISTICS.

Problem	Total Time as Reported by				Perturbation
Instance	TS-B	SA-B	GLS	SA-KN	Method
beas50-1	14	19	-	-	0.65
beas50-2	16	20	-	-	0.56
beas50-3	17	21	-	-	1.3
beas50-4	16	21	-	-	1.05
beas50-5	16	20	-	-	1.8
beas50-6	16	22	-	-	1.1
beas50-7	17	22	-	-	1.7
beas50-8	17	22	-	-	1.7
beas50-9	17	22	-	-	1.6
beas50-10	17	21	-	-	1
beas100-1	34	31	-	-	1.4
beas100-2	35	34	-	-	1.3
beas100-3	37	34	-	-	3.1
beas100-4	33	33	-	-	1.5
beas100-5	36	33	-	-	2.3
beas100-6	36	34	-	-	1.3
beas100-7	36	32	-	-	1.1
beas100-8	36	31	-	-	3.5
beas100-9	35	32	-	-	1.3
beas100-10	38	36	-	-	2.9
beas250-1	238	226	-	-	2.4
beas250-2	239	226	-	-	4.4
beas250-3	254	240	-	-	4.1
beas250-4	234	218	-	-	2.7
beas250-5	245	232	-	-	4.2
beas250-6	240	221	-	-	3.3
beas250-7	250	232	-	-	2.5
beas250-8	225	212	-	-	4.8
beas250-9	246	229	-	-	2.7
beas250-10	235	218	-	-	3.5
beas500-1	956	1006	-	10	8.79
beas500-2	979	1009	-	10	8.6
beas500-3	987	1030	-	10	7
beas500-4	1003	1061	-	10	11.96
beas500-5	964	1030	-	10	13.1
beas500-6	966	1028	-	10	12.6
beas500-7	952	1014	-	10	12.7
beas500-8	1006	1050	-	10	15.4
beas500-9	954	998	-	10	15.6
beas500-10	971	1012	-	10	13.3

- indicates time not reported

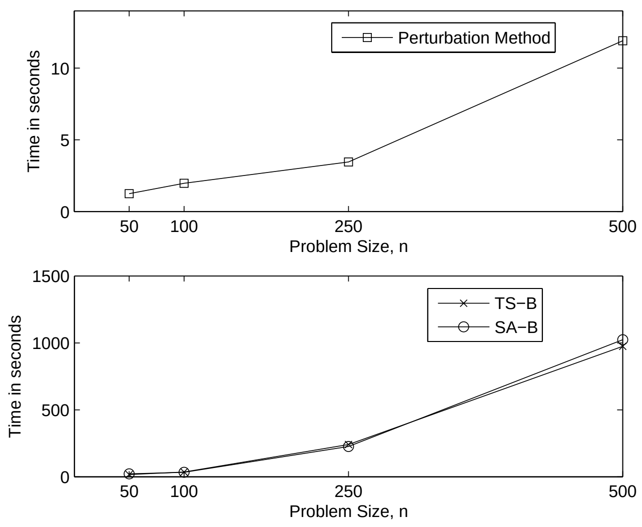

Fig. 1. Comparison of running time with other heuristis.

because of the difference in computer configurations and softwares used. Having factored that in, we still feel that with a modest computer configuration, we were able to produce some good solutions with reasonable computation time. Hence, this method can be used to provide some good starting points from which other heuristics can use to search for global optimal or near-global optimal solutions.

C. Comparison with exact methods

The comparison with other heuristics in the previous section does not justify the performance evaluation of the proposed method thoroughly. It will only be appropriate to compare our method with other exact methods which are available to solve similar problems. However, most of the exact methods proposed for solving the unconstrained binary quadratic programming problems do not report solutions for the standard test problems that we are attempting to solve. Hence, we decided to compare our performance with that of ILOG OPL version 3.7. ILOG OPL uses the CPLEX (version 9.0) engine to solve linear, integer and mixed-integer programming problems. In particular, we used it to solve a linear integer programming formulation of the problem (1) and is shown in (6):

$$\begin{aligned} \max \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}y_{ij} & \\ \text{subject to } y_{ij} &\le x_i \\ y_{ij} &\le x_j \\ y_{ij} &\ge x_i + x_j - 1 \\ y_{ij} &\in \{0,1\} \\ x_i &\in \{0,1\} \\ i = 1,\ldots,n;\ j = 1,\ldots,n; & \end{aligned} \tag{6}$$

The CPLEX MIP solver engine solved each instance of the 50 variable problem to optimality within 3 seconds. The performance for solving the 100 variable problem is shown

TABLE III
COMPARISON OF RUNNING TIME WITH OPL.

Problem	OPL - CPLEX Engine		Perturbation Method	
Instance	Solution	Time(sec)	Solution	Time(sec)
beas100-1	7970	4560	7904	1.4
beas100-2	10681[†]	3591	11036	1.3
beas100-3	12723[‡]	1380	12723	3.1
beas100-4	10368	1546.8	10368	1.5
beas100-5	9083	1369.8	9083	2.3
beas100-6	10210	10602	10122	1.3
beas100-7	10125	1771	10098	1.1
beas100-8	11435	758	11435	3.5
beas100-9	11455	126	11455	1.3
beas100-10	12565	1222	12547	2.9

† recorded at 3591 seconds, program did not terminate for up to 3 hours
‡ recorded at 1380 seconds, program did not terminate for up to 3 hours

TABLE IV
ANALYSIS OF SOLUTION QUALITY FOR 500 VARIABLES

Problem Instance	Best Solution	Average Solution	Avg. % deviation	Standard Deviation	SD % BS[†]
beas500-1	116452	116448.8	0.002	2.08	0.002
beas500-2	128255	128215.1	0.031	54.27	0.042
beas500-3	130812	130809.2	0.002	4.66	0.004
beas500-4	130045	129980.4	0.048	55.64	0.043
beas500-5	125397	125387.8	0.007	6.20	0.005
beas500-6	121118	121080.5	0.092	61.53	0.051
beas500-7	122159	122129.7	0.03	26.01	0.021
beas500-8	123421	123473.1	0.045	45.10	0.037
beas500-9	120616	120577.6	0.032	44.01	0.036
beas500-10	130608	130591.4	0.022	33.49	0.026

†-Standard Deviation as % of best known solution

and compared to that of the proposed algorithm in Table III. However, the program was not able to read the data for the 250 variable problem from a text file and so we are unable to show the corresponding computational results for the larger instances. The largest computation time to achieve optimality is seen in the beas100-6 instance. This can be attributed to the highest number of non-zero entries (1038) in the matrix $\mathbf{A}$ for this particular instance. Another problem instance, beas100-2, with 976 non-zero entries does not solve to optimality. This could be due to the fact that the CPLEX MIP solver engine depends on the matrix structure to efficiently solve the problem. A comparison of computation time between OPL and that of our method is shown in Figure 2. Since the observed values of computation time cover a wide range, the time comparison has been made on a logarithmic scale. Though the CPLEX MIP engine solves the problem to global optimality

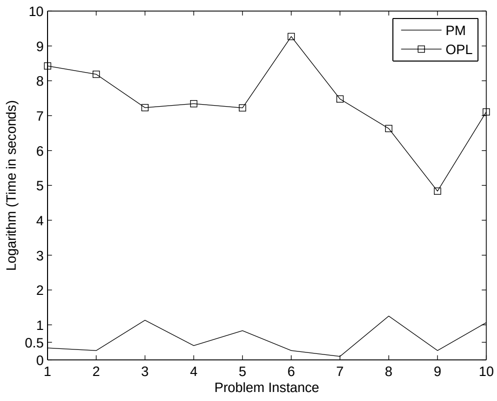

Fig. 2. Comparison of running time with OPL.

most of the time, it is not as good as the perturbation approach in terms of computation time needed to solve the instances. Moreover, OPL has difficulty handling problem instances with more than 100 variables.

D. Effect of random starting points

As mentioned earlier, we use 10 different random starting points from which the algorithm is run. Starting from different points gives us a set of solutions from which the best is selected. Here, the solutions that have been obtained from 10 different random starting points for each instance of the 500 variable problem are analyzed. The average solution quality and its associated statistics are presented in Table IV. The average of the objective values of the solutions obtained and their standard deviations are also provided for each instance. For each and every solution obtained from a random starting point, the average percentage deviation from the best solution, $100(1 - \frac{f_{PM}}{f_{best}})$, is given. Though the method is started from different points, on the average, both the percentage deviation and the standard deviation are around 0.03% of the best known solution. This is within an acceptable range, considering the problem being solved is one of the larger instances (500 variables) in the OR Library.

VII. CONCLUSIONS

In this paper, a perturbation based search method has been proposed to solve the unconstrained binary quadratic programming problem. In order to assess the performance of the proposed algorithm, standard test problems were solved and the solutions are reported. The results obtained were also compared with that obtained by some of the solution methods in the literature. We found that the proposed algorithm is computationally inexpensive and produces some best known solutions. On the average, the results reported were only 0.11% less than the best known solution.

In order to enhance the method further, some heuristic techniques can be incorporated along with the existing method. This would help to improve the solution quality. Wider ranges and different combinations of the parameters used in the method can be tested to see if they affect the solution obtained. Since we have found that the algorithm implemented here is effective for solving the unconstrained binary quadratic programming problem, its application in other areas of solving similar optimization problems could also be explored.

ACKNOWLEDGMENT

This research was supported in part by the Academic Research Council, Ministry of Education, Singapore under the following grants: R-266-000-033-112 and R-266-000-033-133.

REFERENCES

[1] C. R. Cassady, L. M. Maillart, and S. Salman, “Ranking sports teams: A customizable quadratic assignment approach,” *Interfaces*, vol. 35, no. 6, pp. 497–510, November 2005.

[2] A. T. Phillips and J. B. Rosen, “A quadratic assignment formulation of the molecular conformation problem,” *Journal of Global Optimization*, vol. 4, no. 2, pp. 229–241, March 1994.
[3] P. M. Pardalos and G. P. Rodgers, “A branch and bound algorithm for the maximum clique problem,” *Computers and Operations Research*, vol. 19, no. 5, pp. 363–375, July 1992.
[4] K. G. Palubeckis, “A tight lower bound for a special case of quadratic 0–1 programming,” *Computing*, vol. 77, no. 2, pp. 131–145, April 2006.
[5] P. M. Pardalos and G. P. Rodgers, “Computational aspects of a branch and bound algorithm for quadratic zero-one programming,” *Computing*, vol. 45, no. 2, pp. 131–144, June 1990.
[6] H. van Maaren and J. P. Warners, “Bounds and fast approximation algorithms for binary quadratic optimization problems with application to max 2sat,” *Discrete Applied Mathematics*, vol. 107, no. 1-3, pp. 225–239, December 2000.
[7] C. Helmberg and F. Rendl, “Solving quadratic (0,1)-problems by semidefinite programs and cutting planes,” *Mathematical Programming*, vol. 82, no. 3, pp. 291–315, August 1998.
[8] F. Glover, G. A. Kochenberger, and B. Alidaee, “Adaptive memory tabu search for binary quadratic programs,” *Management Science*, vol. 44, no. 3, pp. 336–345, March 1998.
[9] M. Lewis, B. Alidaee, and G. Kochenberger, “Using xQx to model and solve the uncapacitated task allocation problem,” *Operations Research Letters*, vol. 33, no. 2, p. 176 182, March 2005.
[10] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego, “An unconstrained quadratic binary programming approach to the vertex coloring problem,” *Annals of Operations Research*, vol. 139, no. 1, pp. 229–241, October 2005.
[11] K. Katayama and H. Narihisa, “Performance of simulated annealing-based heuristic for the unconstrained binary quadratic problem,” *European Journal of Operations Research*, vol. 134, no. 1, pp. 103–119, October 2001.
[12] Z. Drezner, “A new genetic algorithm for the quadratic assignment problem,” *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 320–330, Summer 2003.
[13] A. Lodi, K. Allemand, and T. M. Liebling, “An evolutionary heuristic for quadratic 0–1 programming,” *European Journal of Operations Research*, vol. 119, no. 3, pp. 662–670, December 1999.
[14] C. Dang and L. Xu, “Barrier function method for the nonconvex quadratic programming problem with box constraints,” *Journal of Global Optimization*, vol. 18, no. 2, pp. 165–188, October 2000.
[15] M. S. Bazaraa and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*. Singapore: John Wiley & Sons, 1990, pp.252–330.
[16] J. E. Beasley, “OR-library: Distributing test problems by electronic mail,” *The Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, November 1990.
[17] J.E.Beasley, “Heuristic algorithms for the unconstrained binary quadratic programming problem,” Management School, Imperial College, London, UK, Tech. Rep., December 1998.
[18] P. Merz and B. Freisleben, “Genetic algorithms for binary quadratic programming,” in *GECCO-1999: Proceedings of the Genetic and Evolutionary Computation Conference*, California, 1999, pp. 417–424.
[19] K. Katayama and H. Narihisa, “Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem,” *European Journal of Operational Research*, vol. 134, no. 1, pp. 103–119, October 2001.

Muthu Solayappan is currently a Ph.D student at the Department of Industrial and Systems Engineering, National University of Singapore. He obtained his M.S in Industrial and Systems Engineering from University of Florida. His research interests are in the areas of interior point methods in nonlinear programming and its applications in computational biology.

Kien Ming Ng is currently an assistant professor at the Department of Industrial and Systems Engineering, National University of Singapore. He obtained his PhD in Management Science and Engineering from Stanford University. His research interests are in optimization, numerical algorithms and operations research applications in military & logistics.

Kim Leng Poh is currently an associate professor at the Department of Industrial & Systems Engineering, and Deputy Director of Temasek Defense Systems Institute at the National University of Singapore. He received his PhD in Engineering-Economic Systems from Stanford University. His current research and consulting interests include decision analysis, investments & risk analysis, automated decision making under uncertainty & resource constraints and large-scale optimization.