Arab Journal of Administrative Sciences

2000, Vol. 7, No. 3, 533 - 550

# A GENETIC ALGORITHM FOR QUADRATIC PSEUDO-BOOLEAN FUNCTION

*Jafar M. Ali*

*Merza Hasan*

*Talal AlKhamis*

Kuwait University

## Key words

***heuristics; genetic algorithm; quadratic function.***

## Abstract

*The applications of genetic algorithm (GA) have grown rapidly in recent years and are being used in several disciplines ranging from industrial to financial fields. The characteristic and the nature of the problem itself render such a procedure. A meta-heuristic procedure genetic algorithm (GA), for the unconstrained quadratic Pseudo-Boolean function is developed. Several parameters were tested on different problem sizes to observe its final performance in terms of solution quality and computational effort. Computational results also demonstrate some discrepancies in the final solutions with different parameters. Finally, the proposed heuristics is capable of solving hard instance problems with a new quality solution in less computational time.*

## Introduction

We considered the unconstrained 0-1 Pseudo-Boolean quadratic problem (QP) of the following form

$$minf(x) = \tfrac{1}{2}x^T Qx + c^T x,$$

where Q = [q$ij$] is an n x n symmetric rational matrix; c is a column vector of n constants, x is a column vector of n binary (0 or 1) variables and $x^T$ is the transpose of $x$.

An equivalent formulation can be written without the linear term Paradalos and Rodgers (1990),

$$minf(x) = x^T Ax \qquad \text{(QP)}$$

where $a_{ii} = (c_i + q_{ii})/2$ and for $i \neq j, a_{ij} = q_{ij}/2$

Problem QP plays a very important theoretical and practical role in combinatorial optimization. From a practical point of view, a large number of industrial problems can be formu-

Submitted October 1998, Accepted January 2000.

Financed by Kuwait University Under Grant CA002.

lated as QP; such as financial analysis problems (Kraup and Pruzan, 1978), Computer Aided Design (CAD) problems (Koulamas *et al*, 1994), models of message traffic management (Gallo *et al*, 1980), vertex packing problems, maximum cut problems, quadratic assignment and set partitioning problems, to name a few. From a theoretical point of view, QP has an interesting connection with other fields of optimization, such as graph theory and other branches of continuous optimization. Using graph theory notations, we can associate a graph $Gf$ (V,E) with $f$ as follows (Chakradhar and Bushnell, 1992):

$$V = \{x_1, x_2, ...., x_n\},$$

$$(x_i, x_j) \in E \Longleftrightarrow Q_{ij} \neq 0 (i \in V, j \in V)$$

where V is the vertex set and E is the edge set of the graph. Hammer (1965) has proved the equivalence between quadratic 0-1 optimization and the problem of finding a maximum cut in an oriented weighted graph. Also, Hammer *et al* (1984) has shown the equivalence between QP and the problem of finding a maximum weight stable set in a graph (vertex packing).

Problem QP is considered NP-hard. Garey and Johnson (1979) and other approaches have proposed to obtain good lower bounds on its optimal value. In the literature, many bounding methods have been presented for QP and its variants. All of these bounds can, in principle, be obtained by solving some linear programming problems through linearizing QP. When Q is an upper triangular matrix of constants, problem QP can be written in the following form:

$$minf(x) = \sum_{1 \leq i < J \leq n} q_{ij} x_i x_j + \sum_{i=1}^{n} q_i x_i \quad \overline{(\mathrm{QP})}$$

A standard technique to linearize $\overline{(\mathrm{QP})}$ is to introduce a new variable that takes the value $x_i x_j$ for every $x \neq \{0, 1\}$ where $(1 \leq i < J \leq n)$ We thus obtain the following equivalent linear 0-1 program (Boros *et al*, 1990).

$$minf(x, \gamma) = \sum_{1 \leq i < J \leq n} q_{ij} \gamma_{ij} + \sum_{i=1}^{n} q_i x_i$$

Subject to

$$\left.\begin{aligned} \gamma_{ij} &\geq 0 \\ x_i - \gamma_{ij} &\geq 0 \\ x_j - \gamma_{ij} &\geq 0 \\ 1 - x_i - x_j + \gamma_{ij} &\geq 0 \end{aligned}\right\} \quad 1 \leq i < j \leq n$$

$$x_i \in \{0, 1\}, \qquad 1 \leq i \leq n.$$

Hammer *et al* (1984) present three possible approaches for obtaining strong bounds on the optimal value of QP: roof dual, and complementation and linearization. Boros *et al* (1990) generalized these approaches and studied their mutual relationships. In another paper, Boros *et al* (1992) presented a new lower bound for QP. They showed that this lower bound could be computed by solving a linear programming problem of polynomial size in the number of variables.

Adams *et al* (1990) considered the problem of maximizing a general Pseudo-Boolean function. They formulated the problem as a constrained 0-1 polynomial program and showed how to compute an upper bound on the maximum objective function value through the construction of a lagrangean dual. This lagrangean dual yields an optimal objective value equal to the roof dual value for 0-1 polynomial programming problems.

Using these bounding techniques, a number of researchers have presented a branch and bound based algorithm to solve problem QP. Williams (1985) presented an algorithm to solve QP based on a branch and bound method. The algorithm starts with a reduction process to obtain a good initial solution, and then uses the roof dual linear program as described by Hammer *et al* (1984) to provide bounds. Barahona *et al* (1989) developed an approach where QP is reduced to a max-cut problem and then uses a partial description of the cut polytope to solve the problem using linear programming techniques. They used branch and bound if no integer solution was found by their cutting plane algorithm. Billionnet and Sutter (1994) present a branch and bound algorithm for minimizing QP. At each node of the search tree, the lower bound (bi) is computed in three phases and is equal to $b_1 + b_2 + b_3$. Computation of $b_1$ is based upon roof duality, $b_2$ is computed using the characterization of some positive quadratic posiforms associated with the directed cycles of the implication graph of Aspvall *et al* (1979), and b3 is computed by searching in a posiform of degree 4, some sub-functions which cannot be equal to zero. These sub-functions are found by using the notion of implication between literals. Pardalos and Rodgers (1990) proposed a simple but powerful technique of fixing variables in a branch and bound algorithm for solving QP. The algorithm incorporates dynamic preprocessing techniques of forcing variables and heuristics to obtain good starting solution.

As stated earlier, the problem of minimizing QP is known to be NP-hard. However, there are three special cases where the problem is known to be solvable in polynomial time. Picard and Ratliff (1974) have given a polynomial time algorithm to solving the special case when all the elements of the matrix $[q_{ij}]$ are nonpositive. Barahona (1986) has given a polynomial time algorithm for the case when the graph Gf is series parallel. Crama *et al* (1988) generalized this result and proposed a polynomial time algorithm for the case when the graph $Gf$ is of bounded tree width. Chakradhar and Bushnell (1992) present a new class of

quadratic 0-1 programming cases solvable in O($n$) time where n is the number of variables in the Pseudo-Boolean quadratic function. They established a direct and constructive relationship between quadratic Pseudo-Boolean functions and logic circuits. They showed that if the graph $Gf$ is transformable into a combinatorial circuit of inverters and 2-input AND, OR NAND and NOR logic gates then the minimum of f can be obtained in linear time. They used a novel modeling technique to transform $Gf$ into a logic circuit. The minimum of $f$ was determined by identifying the primary input signals of the circuit and by performing logic simulation of an arbitrary set of 0-1 values for these input signals.

Although exact methods and bounding techniques for solving QP exist in literature, they can only solve small to medium-size problems with acceptable CPU time. These methods require extensive computations and large CPU time to solve large-scale problems. In this paper, we will introduce the structural procedure of Genetic Algorithm (GA) to solve QP, and present extensive computational results of its performance for a new set of difficult test problems. Based on existing literature, one cannot find complete heuristic procedures, including their computational results, for solving QP. Therefore, we proposed three meta-heuristic algorithms that have been proven to solve complex combinatorial optimization problems, and many researchers have provided good solutions with excellent computational results for problems that are considered to be NP-hard. For general presentation of the meta-heuristic algorithms with various successful applications to combinatorial optimization problems, the reader may refer to Colorni *et al* (1996).

This paper is organized as follows: in the next section we will introduce a mechanism to generate an alternative solution from the current solution to be imbedded in the heuristic procedure. The modification and implementation of the meta-heuristic procedure GA is presented in section 3. Computational comparisons on a set of 45 different test problems considered hard to be solved by other procedures (see Alkhamis *et al*, 1998) is provided in section 4. And Finally, our conclusions and perspectives on future research are presented in section 5.

## Solution Interchange and Data Structure

Before we start our GA procedure, an initial solution should be generated either randomly or by applying a simple construction procedure as a

seed for our population construction. For a given quadratic polynomial function $f(x)$, a solution $S$ can be characterized by a string $h$, of length $n$, and of binary values representing the variables $X's$. Each element of the string $h$ represents a variable $x_i$ where $x_i \in \{0, 1\}$, with an objective function value, $W(s) = f(X)$. A generation mechanism generates solutions by a $\lambda$ interchange in $h$. The solution space, $N_\lambda(S)$, is a set of all generated solutions $S'$ by the generation mechanism from $S$, for a given integer $\lambda$. For $\lambda = 1$, a 1-interchange has been applied by simply setting a variable $x_i$ of the string $h$ to $x_i = 1\text{-}x_i$ The length of $\lambda$ is related to the number of changes of the variables $x_i$ 's in the string $h$ and it is an important concept for the GA crossover procedure that will be discussed later.

To reduce the computational time associated with any changes in the string $h$, a data structure has been developed that will compute only the changes in the string $h$. For example, consider a 1-interchanged procedure applied to the variable $x_k$, where the coefficient $q_{ij}$ is stored as an $n \times n$ symmetrical matrix. Variable $x_k$ has three possible interchanges: from 0 to 1, from 1 to 0, or no change. In the case of no change, the procedure will not be implemented. For the change we defined the variable sign as follows: set sign = 1 when $x_k$ is changed from 0 to 1 and sign = -1 otherwise. Then the new $W(S')$ will be computed as follows:

*For* $i = 1$ *to* $n$ *Do*

$$W(S') = W(S) + (2 \text{ sign } q_{ik}x_i)$$

end Do

$$W(S') = W(S') - q_{kk}$$

The above procedure for computing $W(S')$ is implemented for each interchange and computed in O($n$) time. Consequently, the interchange size and its computational requirement increase with the value of $\lambda$.

## Genetic Algorithm

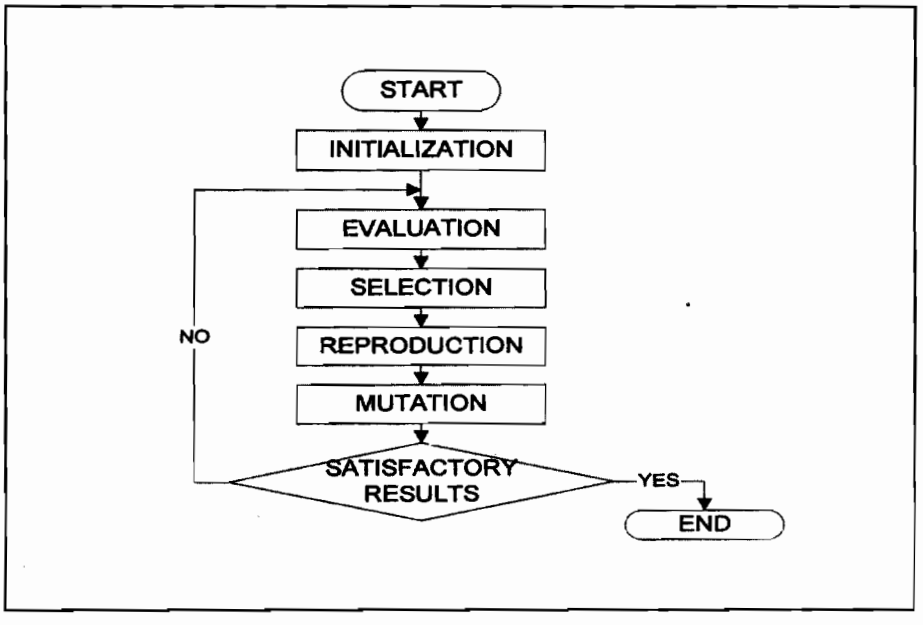
A genetic algorithm (GA) procedure attempts to solve real problems by mimicking the mechanism of natural evaluation process. Currently, GA has been applied to a variety of computational optimization problems (Reeves 1993). John Holland (1975) was the first to introduce the theoretical foundations of GA and how to apply it artificial systems.

A genetic algorithm (GA) can be understood as an "intelligent" search algorithm, which can be applied to a variety of combinatorial optimization problems (Reeves 1993). The theoretical foundations of GAs were originally developed by Holland (1975). The idea of GAs is based on the evolutionary process of biological organ-

isms in nature. During the course of evolution, nature evaluations, and natural populations evolve according to the principles of natural selection and "survival of the fittest". Species that are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst species that are less fit will be eliminated. This means that the genes from the highly fit species will be spread to an increasing number of individuals in each successive generation. The combination of characteristics from highly adapted ancestors may produce even more fit offspring. In this way, species evolve to become more and more well adapted to their environment.

A GA simulates these processes by taking an initial population and applying genetic operators in each reproduction. In optimization terms, each individual in the population is encoded into a string or chromosome which represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or solutions are given opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit individuals. This pro-

**Figure 1**
**Genetic Algorithm Cycle**



duces a new "offspring" solution (i.e. children), which shares some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the strings. The offspring can either replace the whole population (generational approach) or replace less fit individuals (steady-state approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found. The basic steps of a simple GA are shown below. A more comprehensive overview of GAs can be found in Beasley *et al* (1993), Goldberg (1989), and Reeves (1993).

## The GA-based heuristics

The component of the GA described in the previous section will be applied to the QP problem as follows.

### *Representation*

The first step in a genetic algorithm is to determine the internal representation of the genes. The most common representation of the genes uses a binary coding where each gene can have either 1 or 0. Therefore, each string $h$ consists of 0's and 1's and used a n-bit binary string as the chromosome structure where $n$ is the number of variables $X$ in the QP problem.

$h[1], h[2], ....., h[n]$ where $h[k]$ either 0 or 1 and $1 \leq k \leq n$

### *Initialization and Evaluation*

An initial population is created randomly where the gene in each chromosome consists of randomly assigned values of either 0 or 1. Each chromosome will be assigned a fitness value, which will be used in the selection process. To determine the fitness value for each chromosome, an objective function $W(S)$ for the problem should be evaluated. To strengthen the generated initial population, a local search (LS) is applied for each chromosome before entering the initial population. For each chromosome, $S$, a neighborhood $S'$ is chosen from $N_1(s)$, using 1-interchange procedure and the difference in the objective function values $\Delta = W(S') - W(S)$ is computed. The acceptance criterion considers to be admissible if $\Delta < 0$. Our LS uses the systematic search strategy that selects the best admissible neighbor to replace the current solution (BA selection strategies). If $\Delta \geq 0$ for n number of iterations, the algorithm is terminated.

### *Parent Selection*

The selection mechanism elects parents from the population $(POP)$ for the genetic manipulation process, and reproduction, by allocating reproductive opportunities to each individual.

Binary tournament selection (i.e. B = 2) has been chosen as the method of parent selection process due to its efficiency in implementation and generating better solution quality.

We chose the binary tournament selection (i.e. B = 2) as the method of parent selection because it can be implemented very efficiently and generate better solution quality.

Parent selection is the task of assigning reproductive opportunities to each individual in the population (POP). There are a number of widely used methods including proportionate selection and tournament selection. The proportionate method calculates the probabilities of individuals selected in proportion to their fitness and selects individuals for mating based on this probability distribution. The tournament selection method works by forming two or more pools of individuals, each consisting of B individuals drawn from the population randomly. Two individuals from each tournament are mated and the best fitness is chosen. Using a larger value for B has the effect of increasing selection pressure on the more fit individuals.

We chose the binary tournament selection (i.e. B = 2) as the method of parent selection because it can be implemented very efficiently and generate better solution quality.

*Reproduction*

After the selection of parents a reproduction phase occurs wherein selected parents mate and produce offspring, thereby encompassing the next generation. The selected parents' chromosomes are recombined using crossover and mutation mechanisms.

## Crossover

The most basic form of crossover operation is the one-point crossover where each selected parent is randomly cut into two parts of equal length to reproduce two "head" segments and two "tail" segments. The "tail" segments are swapped to produce new offspring. By swapping the chromosomes, the offspring will inherit some genes from each parent. In this work, the length of segments, which consists of finite number of genes, is called crossover length. The one-point crossover is formally defined as:

Let $h_1$ and $h_2$ be the parent strings $h_1[1], ..., h_1[n]$ and $h_2[1], ..., h_2[n]$ respectively. Let us assume the crossover points at position $r$, where $1 \leq r < n - 1$. Then the child strings $C_1$ and $C_2$ are:

$$C_1 := h_1[1], h_1[2], ..., h_1[r], h_2[r+1], h_2[r+2], h_1[r+3], ...h_1[n]$$

$$C_2 := h_2[1], h_2[2], ..., h_2[r], h_1[r+1], h_1[r+2], h_2[r+3], ...h_2[n]$$

Another type of crossover mechanism, which is generally consid-

ered better, is the two-point crossover, which is formally defined as:

Let $h_1$ and $h_2$ be the parent strings $h_1[1], ..., h_1[n]$ and $h_2[1], ..., h_2[n]$ respectively. Let us assume the length of crossing is two bits, which generates two crossover points at position $r$ and $k$ randomly, where $1 \leq r < n-1$, and $1 \leq k < n-1$. Then the child strings $C_1$ and $C_2$ are:

$$C_1 := h_1[1], h_1[2], ..., h_1[r], h_2[k+1], h_2[k+2], h_1[k+3], ...h_1[n]$$

$$C_2 := h_2[1], h_2[2], ..., h_2[r], h_1[r+1], h_1[r+2], h_2[r+3], ...h_2[n]$$

The cross point operator and the crossing length are important to generate a good child different from its parent. That crossover can be computed in O $(n)$ time at each iteration.

## Mutation

After the *crossover operation, mutation* process then is applied for each child, which it considered as a "background" operator that provides a small amount of random search. Mutation is implemented by occasionally inverting randomly chosen gene $(h_i)$ in the chromosome with a small probability.

$$C = h[1], h[2], ..., -h[k], ..., h_1[n]$$

Even though, mutation is used with small probability, it is considered more productive than the crossover operation as the population converges (Davis 1991). The purpose of mutation is to reintroduce any fortuitously lost gene values due to premature convergence and thereby expanding the search space. In addition, it might introduce new gene information.

We follow Back's (1993) mutation rate of $1/n$ as a lower bound on the optimal mutation rate, where n is the number of genes in the chromosome. This lower bound is equivalent to mutating one randomly chosen bit per string after v crossing iterations.

### *Population replacement model*

In this phase, the worst member in the population has been replaced with the new generated child solution that has a better fitness value. Nonetheless this will increase the computational time. The best solutions are always kept in the population and the newly generated solutions are immediately available for selection and reproduction. In addition, there is a high probability that a duplicate child, that has the exact solution structure to an existing *POP* solution structures, will enter the population where a population could come to consist of identical *POP* solutions. A mechanism is implemented to not permit duplicate solutions from entering the population.

## Computational Experience

In the absence of any standard benchmark problems for this type of problem (especially when the optimal solution is not known), it was decided to compare the performance of GA with the heuristic developed by Al-Khamis *et al* (1998). AlKhamis *et al* (1998) and presented computational results for QP where the coefficient of Q and c are integral and uniformly distributed in the intervals $[Q_{min}, Q_{max}]$ and $[C_{min}, C_{max}]$ respectively; where $-Q_{min} = Q_{max} = -C_{min} = C_{max} = N$ with density D. The density D is the percentage probability that any particular coefficient is not zero, and N is the largest possible value for |qij|. The density D along with the number of variables in the problem, n, is the major factors of the problem difficulty. They consider test problems with $-Q_{min} = Q_{max} = -C_{min} = C_{max} = 100$ and density ranging from 10% to 100% and show their performance on a microcomputer.

The main theme of this work is to observe the performance of the GA on the set problems given by AlKhamis *et al* (1998). The experiment calls for the generation of random integer entries in the interval $[-Q_{max}, Q_{max}]$. Forty-five problems for each of the profiles (n, D, and $Q_{max}$) were solved (table 1). The first column of table 1 represents the problem size n of various types ranging from 40 to 100 with $-Q_{max}$ and $Q_{max} = 100$. We also reported the seed for each problem to generate the same $q_{ij}$ matrix using AlKhamis *et al* (1998) code for further comparison, then followed by the best solution that has been reported by AlKhamis *et al* (1998) using simulated annealing procedure. The reported computational time for the GA procedure is the total running time excluding the construction of the initial population. For each test problem, we reported the relative percentage deviation (%DEV) over the best solution heuristic developed by AlKhamis *et al* (1997). The generated test cases are considered to be difficult to solve, and their optimal solution cannot be obtained in a reasonable amount of computation time by applying the procedures proposed in the literature. The final solution result is presented in table 1.

## Parameter Setting:

The most difficult and time-consuming aspect in the successful implementation of genetic algorithms is to find good parameter settings. Here we point out the trade offs that arise:

i) Increasing the crossover length increases the recombination of building bocks, but it also increases the disruption of good string.

**Table 1**
**Comparison between GA final solution procedure and AlKhamis *et al* heuristic**

| N | D% | Seed | est Solution | CPU (Sec) | GA Solution | CPU (SEC) | Best Procedure | Deviation of GA from best Heuristic (DEV%) |
|---|---|---|---|---|---|---|---|---|
| 40 | 90 | 37 | -3762 | 0.930 | -3762 | 4.505 | | 0.000 |
| 40 | 100 | 42 | -3604 | 0.875 | -3604 | 4.451 | | 0.000 |
| 50 | 70 | 11 | -4426 | 2.195 | -4426 | 4.725 | | 0.000 |
| 50 | 80 | 13 | -4961 | 2.250 | -4961 | 4.725 | | 0.000 |
| 50 | 90 | 23 | -4774 | 2.258 | -4774 | 4.725 | | 0.000 |
| 50 | 100 | 27 | -5414 | 2.250 | -5414 | 4.780 | | 0.000 |
| 60 | 50 | 10 | -4737 | 4.563 | -4737 | 4.945 | | 0.000 |
| 60 | 60 | 14 | -4937 | 4.563 | -5337 | 5.220 | | -0.081 |
| 60 | 70 | 17 | -6099 | 4.609 | -6099 | 5.000 | | 0.000 |
| 60 | 80 | 22 | -7600 | 4.617 | -7600 | 4.945 | | 0.000 |
| 60 | 90 | 34 | -7955 | 4.609 | -7955 | 5.000 | | 0.000 |
| 60 | 100 | 60 | -7084 | 4.563 | -7084 | 4.945 | | 0.000 |
| 70 | 40 | 11 | -6461 | 8.406 | -6461 | 5.275 | | 0.000 |
| 70 | 50 | 13 | -6417 | 8.461 | -6417 | 5.275 | | 0.000 |
| 70 | 60 | 23 | -6297 | 8.461 | -6297 | 5.275 | | 0.000 |
| 70 | 70 | 27 | -6530 | 8.461 | -6530 | 5.275 | | 0.000 |
| 70 | 80 | 43 | -9436 | 8.461 | -9436 | 5.275 | | 0.000 |
| 70 | 90 | 79 | -8565 | 8.406 | -8566 | 5.330 | GA | -0.001 |
| 70 | 100 | 95 | -7450 | 8.406 | -7450 | 5.275 | | 0.000 |
| 80 | 30 | 12 | -7221 | 14.391 | -7231 | 5.549 | GA | -0.001 |
| 80 | 40 | 15 | -7371 | 14.281 | -7371 | 5.549 | | 0.000 |
| 80 | 50 | 70 | -10079 | 14.391 | -10095 | 5.549 | GA | -0.002 |
| 80 | 60 | 31 | -11478 | 14.344 | -11478 | 5.549 | | 0.000 |
| 80 | 70 | 34 | -11768 | 14.398 | -11768 | 5.549 | | 0.000 |
| 80 | 80 | 8 | -14248 | 14.281 | -14251 | 5.549 | GA | -0.001 |
| 80 | 90 | 80 | -12661 | 14.391 | -12661 | 5.604 | | 0.000 |
| 80 | 100 | 142 | -14447 | 14.336 | -14447 | 5.549 | | 0.000 |
| 90 | 20 | 10 | -8105 | 22.914 | -8105 | 5.879 | | 0.000 |
| 90 | 30 | 12 | -8379 | 22.859 | -8408 | 5.879 | GA | -0.003 |
| 90 | 40 | 15 | -8726 | 22.906 | -8726 | 5.934 | | 0.000 |
| 90 | 50 | 70 | -11537 | 22.914 | -11537 | 5.879 | | 0.000 |
| 90 | 60 | 31 | -9896 | 22.906 | -9911 | 5.934 | GA | -0.002 |
| 90 | 70 | 34 | -11693 | 22.906 | -11853 | 5.879 | GA | -0.014 |
| 90 | 80 | 8 | -15464 | 22.859 | -15464 | 5.879 | | 0.000 |
| 90 | 90 | 80 | -15544 | 22.859 | -15544 | 5.934 | | 0.000 |
| 90 | 100 | 142 | -15561 | 22.969 | -15573 | 5.934 | GA | -0.001 |
| 100 | 20 | 10 | -7120 | 34.672 | -7120 | 6.209 | | 0.000 |
| 100 | 30 | 12 | -10416 | 34.672 | -10460 | 6.264 | | -0.004 |
| 100 | 40 | 15 | -10654 | 34.719 | -10654 | 6.264 | | 0.000 |
| 100 | 50 | 70 | -13837 | 34.727 | -13837 | 6.209 | | 0.000 |
| 100 | 60 | 31 | -11924 | 34.781 | -11924 | 6.264 | | 0.000 |
| 100 | 70 | 34 | -16434 | 34.672 | -16469 | 6.209 | GA | -0.002 |
| 100 | 80 | 8 | -17931 | 34.727 | -17931 | 6.209 | | 0.000 |
| 100 | 90 | 80 | -17176 | 34.781 | -17176 | 6.209 | | 0.000 |
| 100 | 100 | 142 | -17216 | 34.727 | -17216 | 6.209 | | 0.000 |

ii) Increasing the mutation probability tends to transform the genetic search into a random search, but also helps to reintroduce lost genetic material.

iii) Increasing the population size increases the diversity and reduces the probability that the genetic algorithm will prematurely converge to a local optimum, but it also increases the time required for the population to converge.

Two distinct parameter sets have emerged: one has a small population size and relatively large mutation and crossover rate, while the other has a larger population size, but a smaller crossover and mutation rate. Next we will examine the interactive effect of each parameter setting on the final solution.

**Problem size and population size:**

Although the population size is less significant than other main factors, there was a significant interaction between problem size and population size (figure 2). The interaction plot indicates that at smaller populations the percentage deviation for a larger problem size is greater than for a smaller problem size. However, the percentage deviation increases as the population size increases for all problem sizes.

**Problem size and crossover length:**

The interaction between problem size and crossover length (figure 3) is not significant, however the interaction plot indicates that a smaller crossover length is recommended for all problem sizes. For larger problem sizes the crossover length should be less than 2.

**Figure 2**
**Bound sensitivity analysis for various problem and population sizes**

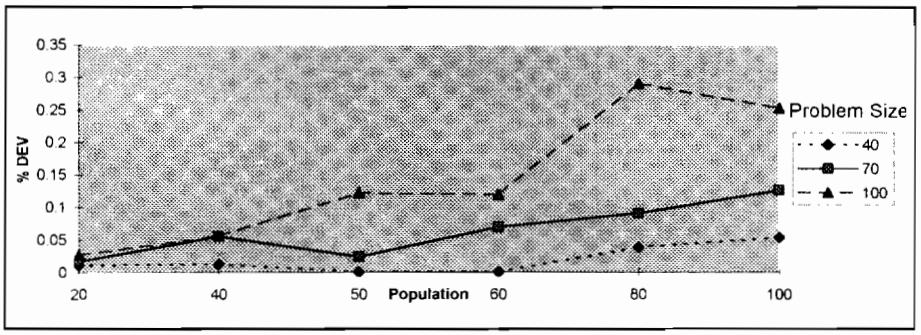
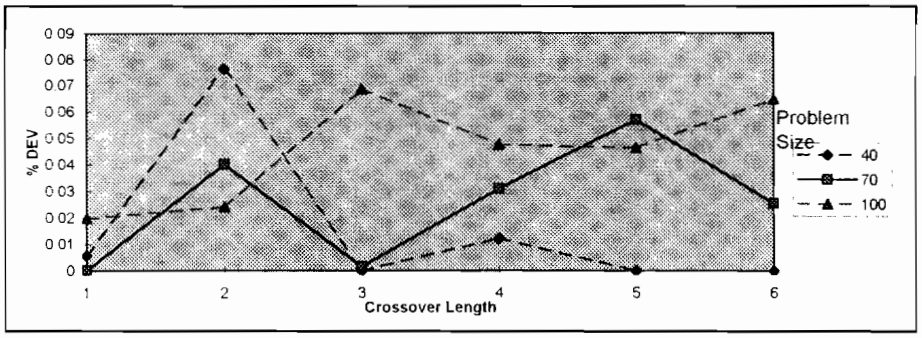


**Figure 3**
**Bound sensitivity analysis for various problem sizes and crossover length**




## Problem size and mutation rate:

There is an extremely significant interaction between problem size and the mutation rate (figure 4). The interaction plot indicates that at lower mutation rate three problem sizes follow almost the same pattern at the mutation rate of 80 and the behavior of the three problem sizes is almost the same with slight difference. As the mutation rate decreases the percentage deviation for small problem size rises suddenly. Therefore, the recommended mutation rate is a rate between 70-85 percent.

Population size and crossover length:

The interaction between population size and crossover length (figure 5) is not significant. The interaction plot indicates that for a lower population size the percentage deviation increases with an increase in crossover length. Hence for a lower population size the disruptive effect of crossover

**Figure 4**
**Bound sensitivity analysis for various problem sizes and mutation rates**

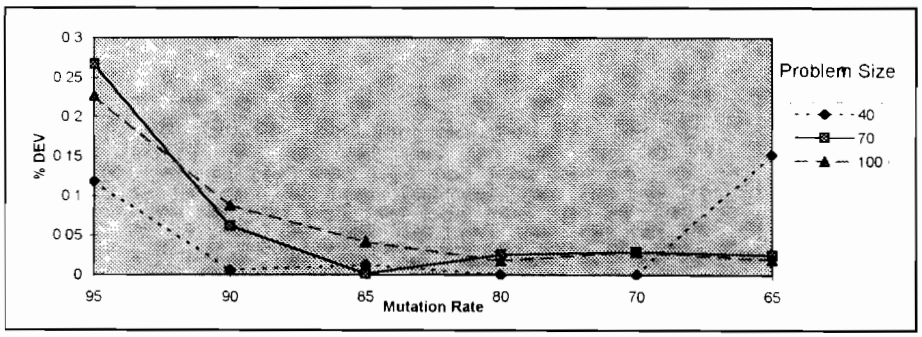
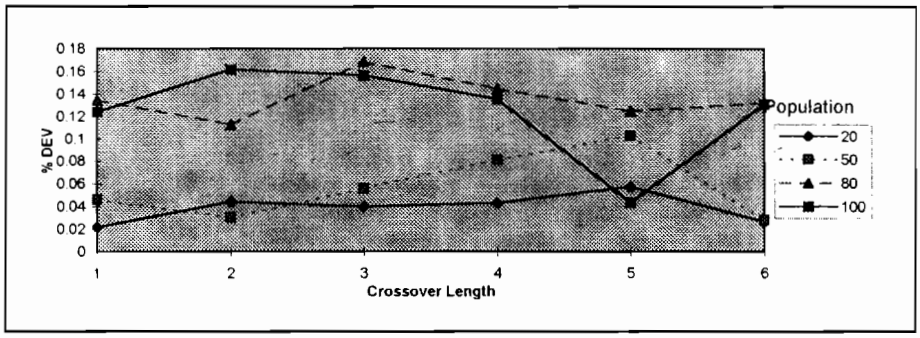


**Figure 5**
**Bound sensitivity analysis for various population sizes and crossover length**




length dominates the recombination. With a population size of 100, the percentage deviation decreases with an increase of crossover length.

## Population size and mutation rate:

There is a very significant interaction between population size and mutation rate (figure 6). The interaction plot indicates the highest mutation rate the lowest percentage deviation for a population size. The lower population sizes perform better with a higher mutation rate. Thus there is some sort of inverse relationship between population size and mutation rate. The interaction plot demonstrates that the mutation rate of 80 percent is equally good for all population size.

## Crossover length and mutation rate:

The interaction between crossover length and mutation rate is significant (figure 7). The interaction plot indi-

**Figure 6**
**Bound sensitivity analysis for various population sizes and mutation rates**

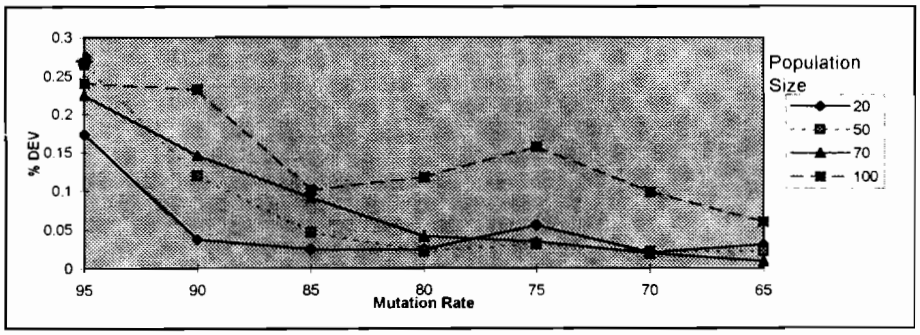
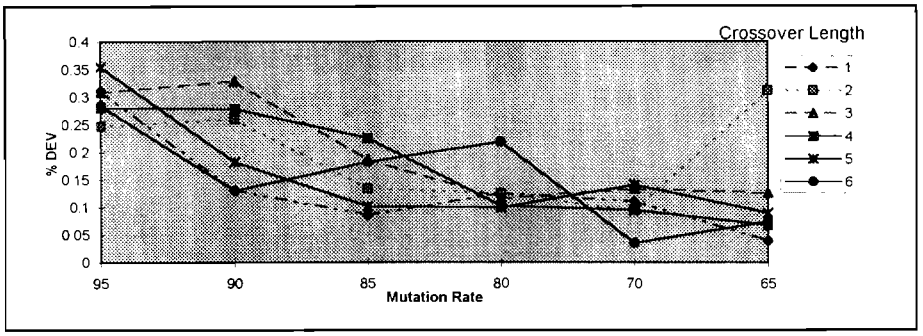


**Figure 7**
**Bound sensitivity analysis for various population sizes and mutation rates**




cates that there is an inverse relation between mutation rates and crossover lengths. At the higher mutation rate and lower crossover length, the algorithm will perform better. However the higher crossover length is good at the higher mutation rate.

## 5. Conclusion

In this paper we investigated different parameter settings for the genetic algorithm implemented to the quadratic unconstraint Boolean function. The successful implementation of GA to the problem under study was demonstrated by addressing the problem of different sizes and comparing the results with the best available heuristic for this kind of problem that has been proposed by AlKhamis *et al* (1998). The result shows that the proposed procedure can generate better solution on average of AlKhamis *et al* (1998) method.

Since the successful implementation of GAs depends upon the careful selection of parameter values, a plot analysis is provided to investigate the interaction between different parameters. The result shows that the crossover length, mutation rate and problem size have a significant impact on the quality of the solution while the population size was found to be less significant.

Although GAs can give better results than the other heuristic for the problem under study, it can also be seen as the problem increases in the size, the computational efforts become lower to generate either the same final solution or lower (table 1). As for the 45 test problems which has been considered difficult to solve optimally our procedure outperformed AlKhamis *et al* (1998) in nine problems. During the study we came across some important issues which could be the basis for future research. These are as follows:

i) incorporating problem specific knowledge, ii) effect on the quality of solution if crossover and/ or mutation rates are varied over the time when the genetic algorithm is running, iii) sensitivity analysis of different parameters can be set quickly and accurately, and iv) hybrid procedure to combine more than one heuristic.

Finally, the main advantage of the GA procedure that it generates good solution in less computational time, where its limitation has mainly on the crossover length and position. However, we can overcome this limitation by integrating GA with other procedure for example Tabu search to regulate the crossing methods.

## References


Adams, W., Billionnet, A., and Sutter, A. 1990. Unconstrained 0-1 Optimization and Lagrangean Relaxation. *Discrete Applied Mathematics*, 29:131-142.

AlKhamis, T, Hasan, M. and Ahmed M. 1998. Simulated Annealing for The Unconstrained Quadratic Pseudo-Boolean Function. *European J. of Operation Research*, 108:641-652.

Aspvall, B., Plass, M., and Tarjan, R. 1979. A Linear Time Algorithm for testing The Truth of Certain Quantified Formulas. *Information Processing Letters*, 8:121-123.

Back, T. 1993. Optimal Mutation Rates in Genetic Search. In S. Forrest, editor, *Proceedings of The Fifth International Conference on Genetic Algorithm*, Pages 2-9, San Mateo, CA, Morgan Kaufmann.

Barahona, F. 1986. A Solvable Case of Quadratic 0-1 Programming. *Discrete Applied Mathematics*, 13:23-26.

Barahona, F., Junger, M. and Reinelt, G. 1989. Experiments in Quadratic 0-1 Programming. *Mathematical Programming*, 44:127-137.

Beasley D., Bull D. and Martin R. 1993. An Overview of Genetic Algorithm: Part I, Fundamentals, *University Computing, 15:58-69.*

Beasley D., Bull D. and Martin R. 1993. An Overview of Genetic Algorithm: Part II, Research Topics. *University Computing*, 15:170-181.

Billionnet, A., and Sutter, A. 1994. Minimization of a Quadratic Pseudo-Boolean Function. *European J. of Operational Research*, 78:106-115.

Boros, E., Carma, Y., and Hammer, P. 1990. Upper-bounds for Quadratic 0-1 Maximization. *Operations Research Letters*, 9:73-79.

Boros, E., Carma, Y., and Hammer, P., 1992. Chvatal Cuts and Odd Cycle Inequalities in Quadratic 0-1 Optimization, SIAM *Journal of Discrete Mathematics*, 5(2):163-177.

Chakradhar, S., and Bushnell, M. 1992. A Solvable Class of Quadratic 0-1

programming. *Discrete Applied Mathematics*, 36:233-251.

Goldberg D. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, New York.

Colorni, A., Dorigo, M., Maffiloi, F., Maniezzo, V., Righini, G. and Trubian, M. 1996. Heuristics from Nature for Hard Combinatorial Problems. *Int. Trans. Opl. Res.*, 3:1-21.

Crama, Y. 1988. *Linerarization Techniques and Concave Extensions in Nonlinear 0-1 Optimization*. RUTCOR Research Report 32-88, Rutgers University, New Brunswick, NJ.

Crama, Y., Hansen, P. and Jaumard, B. 1988. *The Basic Algorithm for Pseudo-Boolean Programming Revisited. Tech.* Rep. RRR # 54-88, Rutgers Center for Operations Research (RUTCOR), Rutgers University, New Brunswick, NJ.

Davis, L. 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold.

Gallo, G., Hammer, P. and Simeone, B. 1980. Quadratic Knapsack Problem. *Mathematical Programming*, 12:132-149.

Garey, M. and Johnson, D. 1979. *Computers and Intractability: A Guide to The Theory of NP - Completeness*, Freeman, San Francisco.

Hammer, P. 1965. Some Network Flow Problems Solved with Pseudo-Boolean programming. *Operations Research*, 13:388-399.

Hammer, P. Hansen, P. and Simeone, B. 1984. Roof Duality, Complementation and Persistency in Quadratic 0-1 optimization. *Mathematical Programming*, 28(2):121-155.

Holland, J. 1975. *Adoption in Natural and Artificial systems*, MIT press.

Koulamas, C., Antony, S. and Jaen, R. 1994. A Survey of Simulated Annealing Applications to Operations Research Problems. *Omega*, 22(1):41-56.

Krarup, J., and Pruzan, P. A. 1978. Computer aided layout design. *Mathematical Programming Study*, 9:75-94.

Paradalos, P. and Rodgers, G. 1990. Computaional Aspects of a Branch and Bound Algorithm for Quadratic Zero-one Programming. *Computing*, 45:131-144.

Picard, J. and Ratliff, H. 1974. Minimum Cuts and Related Problems. *Networks*, 5:357-370.

Reeves, C. 1993. Modern Heuristic Techniques for Combinatorial Problems. Blackwell Scientific.

Williams, A. 1985. *Quadratic 0-1 Programming Using the Roof Dual with Computational Results*. RUTCOR Research Report # 8-85, The State University of New Jersey.

# أساليب الحل للدالة الثنائية باستخدام المورثات

**جعفر محمد علي**
**ميرزا حسين حسن**
**طلال محمد الخميس**
جامعة الكويت

ظهر مؤخرا استخدامات خوارزمية المورثات لحل مشاكل عديدة تعترض قطاعات مختلفة ابتداء من الصناعة وانتهاء بالتمويل والإدارة، حيث أثبتت الدراسات فعالية هذا الأسلوب للوصول إلى الحلول المثلي. وقد تم تطويع الدالة الثنائية كمشكلة يمكن برمجتها على شكل خريطة جينيه حيث تعتمد هذه الخريطة على متغيرات ومؤشرات يجب دراستها بعمق قبل الشروع في الحل وذلك للحصول على حلول ذات جودة عالية وبأقل جهد باستخدام الذاكرة الرئيسية للحاسب الآلي. ويستدل من النتائج والتحاليل المعروضة بأن هذا الأسلوب يمكن الاعتماد عليه بدرجة كبيرة في حل الدالة الثنائية حيث أثبتت النتائج أيضا أن نوعية الحل وسرعة الإنجاز تفوق الأساليب المعروضة في الأوراق العلمية.

**Jafar M. Ali** (Ph.D. in Computer Science, Illinois Institute of Technology, USA, 1995), Assistant Professor of Management Information System, Faculty of College of Administrative Science, Kuwait University. His Area of Interest: Information Systems, Neural Networks, Genetic Algorithms, Decision Support Systems, Business Ethics, and Databases.

**Merza H. Hasan** (Ph.D. from the Management School, Imperial College, UK, 1992), Associate Professor of Operational Research, Kuwait University, Quantitative and Information System Department. Research Interest Areas are Modern Heuristics, Neural Networks, Mathematical Programming and Combinatorial Optimization.

**Talal M. AlKhamis** (Ph.D. in Operations Research, Florida Institute of Technology, U.S.A, 1995), Associate Professor, Statistician Operations Research Department, Kuwait University, His Research Interests Include Discrete Optimization, Metaheuristics, and Stochastic Optimization.