Journal of Heuristics, 8: 197–213, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming

PETER MERZ AND BERND FREISLEBEN
Department of Electrical Engineering and Computer Science (FB 12), University of Siegen, Hölderlinstr. 3, D-57068 Siegen, Germany
email: pmerz@informatik.uni-siegen.de
email: freisleb@informatik.uni-siegen.de

Received July 8, 1999; Revised August 22, 2000

Abstract

In this paper, a greedy heuristic and two local search algorithms, *1-opt* local search and *k-opt* local search, are proposed for the unconstrained binary quadratic programming problem (BQP). These heuristics are well suited for the incorporation into meta-heuristics such as evolutionary algorithms. Their performance is compared for 115 problem instances. All methods are capable of producing high quality solutions in short time. In particular, the greedy heuristic is able to find near optimum solutions a few percent below the best-known solutions, and the local search procedures are sufficient to find the best-known solutions of all problem instances with $n \leq 100$. The *k-opt* local searches even find the best-known solutions for all problems of size $n \leq 250$ and for 11 out of 15 instances of size $n = 500$ in all runs. For larger problems ($n = 500$, 1000, 2500), the heuristics appear to be capable of finding near optimum solutions quickly. Therefore, the proposed heuristics—especially the *k-opt* local search—offer a great potential for the incorporation in more sophisticated meta-heuristics.

Key Words: unconstrained quadratic binary programming, greedy heuristics, local search

1. Introduction

In the *unconstrained binary quadratic programming problem* (BQP), a symmetric rational $n \times n$ matrix $Q = (q_{ij})$ is given, and a binary vector of length n is searched, such that the quantity

$$f(x) = x^t Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \quad x_i \in \{0, 1\} \ \forall i = 1, \ldots, n \tag{1}$$

is maximized. This problem is also known as the (*unconstrained*) *quadratic bivalent programming problem*, the (*unconstrained*) *quadratic zero-one programming problem*, or the (*unconstrained*) *quadratic (pseudo-) Boolean programming problem* (Beasley, 1998).

The BQP is known to be $\mathcal{NP}$-hard (Garey and Johnson, 1979) and has a large number of applications, for example in capital budgeting and financial analysis problems (Laughunn, 1970; McBride and Yormark, 1980), CAD problems (Krarup and Pruzan, 1978), traffic message management problems (Gallo, Hammer, and Simeone, 1980), machine scheduling (Alidaee, Kochenberger, and Ahmadian, 1994), and molecular conformation (Phillips and Rosen, 1994). Furthermore, several other combinatorial optimization problems can be

formulated as a BQP, such as the maximum cut problem, the maximum clique problem, the maximum vertex packing problem and the maximum independent set problem (Ivănescu, 1965; Pardalos and Rodgers, 1992; Pardalos and Xue, 1994).

Several exact methods have been developed to solve the BQP (Pardalos and Rodgers, 1990; Barahona, Jünger, and Reinelt, 1989; Billionnet and Sutter, 1994; Helmberg and Rendl, 1998), but due to the computational complexity of the problem, heuristics have been proposed recently to find solutions to large problem instances, including *tabu search* (Beasley, 1998; Glover, Kochenberger, and Alidaee, 1998a; Glover et al., 1998b), *scatter search* (Amini, Alidaee, and Kochenberger, 1999), *simulated annealing* (Beasley, 1998; Katayama and Narihisa, 1999) and *evolutionary algorithms* (Lodi, Allemand, and Liebling, 1999; Merz and Freisleben, 1999a).

In this paper, a greedy heuristic and two local search algorithms for the unconstrained binary quadratic programming problem (BQP) are presented. The greedy heuristic constructs a feasible solution in n steps where n denotes the problem size, i.e. the number of elements of a solution. In each step, the greedy heuristic selects an element and a value for it by making a most favorable choice. Additionally, a randomized greedy heuristic is described in which random components are added to the deterministic greedy heuristic.

The local search heuristics iteratively search a better solution in the neighborhood of the current solution until no better solution exists and thus a local optimum is reached. The neighborhood of the first local search algorithm—the *1-opt* local search—is defined by the solutions that can be reached by changing a single element in the current solution. The second local search is a *k-opt* local search where k elements in the solution are changed simultaneously. Due to the computational complexity, only a small fraction of the *k-opt* neighborhood is searched by the local search procedure.

The algorithms are tested on a set of 115 problem instances of the BQP. The experiments show that the local search heuristics are able to find optimum or best-known solutions for instances with a problem size of up to 100 easily. The multi-start *k-opt* local search is capable of finding best-known solutions in all runs for all problem instances studied with a problem size less or equal to 250 and for 11 out of 15 instances of size 500.

For larger instances ($n = 500, 1000, 2500$), the algorithms are shown to find near optimum solutions in 10 to 60 seconds on a state-of-the-art workstation: in most cases, the multi-start randomized greedy heuristic, the multi-start *1-opt* local search, and the multi-start *k-opt* local search find solutions within 1%, 0.5%, and 0.1% below the best-known solution, respectively.

The paper is organized as follows. In Section 1, the BQP is described in more detail. Section 2 describes the greedy and local search heuristics for the BQP. The results obtained from various experiments conducted with these algorithms are discussed in Section 3. Section 4 concludes the paper and outlines areas of future research.

2. The binary quadratic programming problem

Throughout this paper, a solution vector x of the BQP is a boolean vector of length n: $x = (x_1, \ldots, x_n) \in X = \{0, 1\}^n$. Thus, the solution space X of the BQP is of size 2^n; it grows exponentially with n.

The density of the matrix Q is sometimes used to characterize a BQP instance. The density $dens(Q)$ is defined as the number of non-zero entries divided by the number of total entries in the matrix Q. Thus, the density varies between 0 and 1.

To demonstrate the relevance in various scenarios, some special cases and a generalization of the BQP are described in the following.

2.1. *Special cases of the BQP*

The BQP has been shown to be a generalization of other combinatorial optimization problems. For example, the maximum clique problem and the maximum independent set problem are special cases of the BQP. Let $G = (V, E)$ be an undirected graph and $\bar{G} = (V, \bar{E})$ be the complement graph of G, where $\bar{E} = \{(i, j) \mid i, j \in V, i \neq j, \text{ and } (i, j) \notin E\}$. Furthermore, let $A_G = (a_{ij})$ be the adjacency matrix of G, I denote the identity matrix, and $T(x) = \{i \mid x_i = 1, i \in V\}$. Then, the *maximum clique problem* is

$$\min_{x \in X} f(x) = x^t Q x, \quad \text{where } Q = A_{\bar{G}} - I. \tag{2}$$

If x^* solves Eq. (2), the maximum clique of G is defined as $C = T(x^*)$ with $|C| = f(x^*)$.

Similarly, the *maximum independent set problem* is

$$\min_{x \in X} f(x) = x^t Q x, \quad \text{where } Q = A - I. \tag{3}$$

If x^* solves Eq. (3), the maximum independent set of G is defined as $S = T(x^*)$ with $|S| = f(x^*)$.

In the *maximum cut problem* the objective function

$$c(x) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i (1 - x_j) \tag{4}$$

has to be maximized, where w_{ij} denotes the weight of the edge $(i, j) \in E$ in the graph $G = (E, V)$ for which the maximum cut is desired. The maximum cut problem can be formulated as a 0/1 quadratic programming problem by assigning:

$$q_{ij} = -\frac{1}{2} w_{ij}, \forall i \neq j, \quad \text{and} \quad q_{ii} = \frac{1}{2} \sum_{j=1}^{n} w_{ij}, \forall i. \tag{5}$$

The maximum cut size $c(x^*)$ is equal to the objective $f(x^*)$ of the corresponding BQP, and the cut itself is $C = \{(i, j) \in E \mid x_i^* = 0 \text{ and } x_j^* = 1\}$.

Another application of the BQP arises in condensed matter physics. The calculation of ground states in *Ising Spin Glasses* is a combinatorial optimization problem in which a configuration of the spins with minimum energy is searched. The energy of an ising spin

glass, in which the spins lie on a two dimensional grid, is given by the Hamiltonian:

$$H(\omega) = -\sum_i \sum_j J_{ij} s_i s_j, \quad s_i, s_j = \pm 1, \tag{6}$$

where J_{ij} denotes the interaction between site i and j on the grid. By setting

$$q_{ij} = 4J_{ij}, \forall i \neq j, \quad \text{and} \quad q_{ii} = -4\sum_{j=1}^{n} J_{ij}, \forall i, \tag{7}$$

the solution of the BQP yields a configuration with minimum energy, where $s_i = 2x_i - 1 \ \forall i$ and $H(\omega) = -f(x) - \sum_i \sum_j J_{ij}$.

2.2. A generalization of the BQP

There is a close relation between binary quadratic programming and *NK*-landscapes as defined by Kauffman (Kauffman and Levin, 1987) to model gene interaction in genetic evolution. This relation is shown in the following. The objective function of the BQP can be decomposed into n functions. The fitness of a BQP solution can thus be rewritten as a sum of functions for each site, called the fitness contributions f_i of component i in the bit vector:

$$f(x) = \sum_{i=1}^{n} f_i\left(x_i, x_{i_1}, \ldots, x_{i_{k(i)}}\right), \tag{8}$$

$$f_i(x) = \sum_{j=1}^{n} q_{ij} x_i x_j \tag{9}$$

Similar to the *NK*-landscapes defined in Kauffman and Levin (1987), the fitness contribution f_i of a site i depends on the gene value x_i and of $k(i)$ other genes $x_{i_1}, \ldots, x_{i_{k(i)}}$. While for *NK*-landscapes $k(i) = K$ is constant for all i, in the BQP $k(i)$ is defined as the number of non-zero entries in the i-th column of matrix Q. The mean $\bar{k}$ of the $k(i)$ is given by $\bar{k} = n \cdot dens(Q)$. The BQP can be viewed as a special class of *NK*-landscapes with a special fitness contribution function f_i. Alternatively, NK landscapes can be seen as a family of more general binary programming problems.

3. Heuristics for the BQP

Heuristics can be divided into construction heuristics and improvement heuristics. The former construct feasible solutions for a given optimization problem from scratch, while the latter take a feasible solution as their input and try to find better solutions by stepwise transformations.

Both types of heuristics can be implemented efficiently and are often capable of producing near optimum solutions for combinatorial optimization problems. If, however, the results

obtained are not sufficient, the algorithms can be combined or incorporated into meta-heuristics such as memetic algorithms (Moscato, 1989), as for example shown in Merz and Freisleben (2000) for the graph bipartitioning problem: A differential greedy heuristic (Battiti and Bertossi, 1998) combined with Kernighan-Lin local search (Kernighan and Lin, 1972) has been shown to be sufficient for structured problems of small or medium sizes. If larger problems have to be solved or the problems are unstructured, a memetic algorithm—a genetic algorithm incorporating the local search *and* the greedy heuristic—has been shown to be more effective. Since greedy and local search heuristics are important (components of) algorithms for combinatorial optimization problems, corresponding approaches for the BQP are desired. Thus, greedy and local search algorithms for the BQP are described in the following. To the best of our knowledge, our proposals are the first algorithms of this kind for the BQP.

3.1. Greedy heuristics

Greedy algorithms are intuitive heuristics in which greedy choices are made to achieve a certain goal. In combinatorial optimization, a solution to a given problem is searched which maximizes or minimizes an objective function. Greedy heuristics are constructive heuristics since they construct feasible solutions for optimization problems from scratch by making the most favorable choice in each step of construction. By adding an element to the (partial) solution which promises to deliver the highest gain, the heuristic acts as a "greedy constructor".

3.1.1. A simple greedy algorithm. In the BQP, a solution is constructed by assigning a binary value to an element in the solution vector x: a zero or a one. The general outline of the greedy algorithm is provided in figure 1. In each step, the heuristic searches for an element k in the solution vector and a value l to assign to it so that a gain function g_k^l is maximized. Afterwards, the value l is assigned to the vector component x_k.

To find an appropriate gain function, we modify the problem by adding a third state: Let $y \in Y = \{0, \frac{1}{2}, 1\}^n$ be a vector in which each component y_i can have three values 0, $\frac{1}{2}$,

```
procedure Greedy(x ∈ X): X;
  begin
    C := {1, ..., n};
    repeat
      find k, l with g_k^l = max_{i∈C, j∈{0,1}} g_i^j;
      x_k := l;
      C := C\{k};
    until C = ∅;
    return x;
  end;
```

Figure 1. A greedy heuristic for the BQP.

and 1. Starting with a vector $\hat{y}$ with $\hat{y}_i = \frac{1}{2}$ for all i, the greedy heuristic can be viewed as a transformation algorithm that transforms $\hat{y}$ into a vector x for which $x_i \in \{0, 1\}$ for all i and thus $x \in X$. The objective of the solution $\hat{y}$ is

$$f(\hat{y}) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} \hat{y}_i \hat{y}_j = \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}. \tag{10}$$

Let $\tilde{y} \in Y$ be a vector that is equal to $y \in Y$ except for component y_k. Then

$$\Delta f(y) = f(\tilde{y}) - f(y) = q_{kk}\left(\tilde{y}_k^2 - y_k^2\right) + 2(\tilde{y}_k - y_k) \sum_{j=1, j\neq k}^{n} q_{kj} x_j. \tag{11}$$

Hence, the gain for changing y_k from 0.5 to 0 (denoted g_k^0) or 1 (denoted g_k^1) can be defined as

$$g_k^1 = \frac{3}{4} q_{kk} + \sum_{j=1, j\neq k}^{n} q_{kj} y_j, \quad \text{and} \quad g_k^0 = -\frac{1}{4} q_{kk} - \sum_{j=1, j\neq k}^{n} q_{kj} y_j. \tag{12}$$

Using this formula, the greedy heuristic as displayed in figure 1 has a runtime complexity of $O(n^3)$ since the calculation of all g_i takes $O(n^2)$ and this has to be done n times before the algorithm terminates. Thus, the greedy heuristic has an expected running time equal to an algorithm that calculates the objective function $f(x)$ for n solutions. However, the greedy algorithm can be implemented more efficiently. If the matrix Q is not stored in a two dimensional array, but instead, for each i in $\{1, \ldots, n\}$ a list of j's for which $q_{ij} \neq 0$ is maintained, the running time is reduced considerably for instances with a low density. To reduce the running time for finding a choice with a highest gain, the following formula can be used to calculate the new gain g_i after y_k has been set to 0 or 1:

$$\Delta g_i^1 = \begin{cases} \frac{1}{2} q_{ik} & \text{if } y_k = 1 \\ -\frac{1}{2} q_{ik} & \text{otherwise} \end{cases} \quad \text{and} \quad \Delta g_i^0 = \begin{cases} -\frac{1}{2} q_{ik} & \text{if } y_k = 1 \\ \frac{1}{2} q_{ik} & \text{otherwise} \end{cases} \tag{13}$$

Thus, once the gains g_i have been calculated, they can be efficiently updated after the component y_k has been set. Since the gains g only change for those i with $q_{ik} \neq 0$, the running time for updating the gains has a complexity in $O(n)$. In figure 2, the pseudo code of the fast greedy heuristic is provided. The running time for the improved heuristic is in $O(n^2)$ since the running time for finding the maximum gain g_k^0 and g_k^1 is in $O(n)$ and has to be performed n times to construct a feasible solution. Thus, the greedy heuristic has the same computational complexity as an algorithm calculating $f(x)$ for a single solution. The running time behavior of the greedy heuristic can be improved even further if additional data structures are used to find the maximum gain in shorter than $O(n)$ time. However, this has no influence on the computational complexity of the algorithm since the time for the initial calculation of the gains dominates over the times for the greedy choices.

```
procedure Greedy(x ∈ X): X;
  begin
    C := {1, ..., n};
    for i := 1 to n do x_i = 1/2;
    calculate gains g_i for all i in {1, ..., n};
    repeat
      find k_0 with g^0_k = max_{i∈C} g^0_i;
      find k_1 with g^1_k = max_{i∈C} g^1_i;
      if g^0_{k_0} > g^1_{k_1} then
        x_{k_0} := 0; C := C\{k_0};
      else
        x_{k_1} := 1 C := C\{k_1};
      endif
      update gains g_i for all i ∈ C;
    until C = ∅;
    return x;
  end;
```

Figure 2. A greedy heuristic for the BQP.

3.1.2. A randomized greedy algorithm. The greedy heuristic described above is deterministic, since it always produces the same solution for a given problem instance. Often it is desired that a construction heuristic produces many different solutions, for example in hybrid algorithms. The above procedure can be randomized as follows.

By making the first choice randomly, i.e. selecting k and l randomly and setting $x_k = l$ in the first step, a random component is incorporated. Furthermore, the deterministic choice among x_{k_0} and x_{k_1} can be replaced by a random choice proportional to the gains $g^0_{k_0}$ and $g^1_{k_1}$: x_{k_0} is set to 0 with probability $\frac{g^0_{k_0}}{g^0_{k_0}+g^1_{k_1}}$ and x_{k_1} is set to 1 with probability $\frac{g^1_{k_1}}{g^0_{k_0}+g^1_{k_1}}$. The pseudo code of the randomized greedy heuristic is provided in figure 3.

3.2. Local search

Local search (LS) algorithms are improvement heuristics that search in the neighborhood of the current solution for a better one until no further improvement can be made, i.e. there is no better solution in the neighborhood of the current solution. Local search algorithms can be categorized by the neighborhoods they consider. For example, the neighborhood of a solution represented by a binary vector can be defined by the solutions that can be obtained by flipping a single or multiple components in the binary vector simultaneously.

3.2.1. 1-opt local search. The simplest form of local search for the BQP is the *1-opt* local search: In each step, a new solution with a higher fitness in the neighborhood of the current solution is searched. The neighborhood of the current solution is defined by the set of solutions that can be reached by flipping a single bit. Hence, the *1-opt* neighborhood contains all solutions with a hamming distance of 1 to the current solution. In our implementation,

```
procedure RandomizedGreedy(x ∈ X): X;
  begin
    C := {1,...,n};
    for i := 1 to n do x_i = 1/2;
    calculate gains g_i for all i in {1,...,n};
    Select k,l randomly and set x_k := l;
    C := C\{k};
    repeat
      find k_0 with g_k^0 = max_{i∈C} g_i^0;
      find k_1 with g_k^1 = max_{i∈C} g_i^1;
      set p = g_{k_0}^0 / (g_{k_0}^0 + g_{k_1}^1);
      if randomNumber[0,1]< p then
        x_{k_0} := 0; C := C\{k_0};
      else
        x_{k_1} := 1; C := C\{k_1};
      endif
      update gains g_i for all i ∈ C;
    until C = ∅;
    return x;
  end;
```

Figure 3. A randomized greedy heuristic for the BQP.

we search for the solution with the highest fitness, i.e. we search for a flip with the highest associated gain in fitness ($g = f_{\text{new}} - f_{\text{old}}$). The gain g_k of flipping bit k in the current solution can be calculated in linear time using the formula

$$g_k = q_{kk}(\bar{x}_k - x_k) + 2 \sum_{i=1, i \neq k}^{n} q_{ik} x_i (\bar{x}_k - x_k), \tag{14}$$

with $\bar{x}_k = 1 - x_k$.

The local search algorithm is given in pseudo code in figure 4. A straightforward implementation of the *1-opt* local search displayed in the figure has a running time of $O(n^2)$ per iteration. Analogous to the greedy heuristic, the efficiency of the algorithm can be increased

```
procedure Local-Search-1-opt(X): X;
  begin
    repeat
      find k with g_k = max_i g_i;
      if g_k > 0 then x_k := 1 − x_k;
    until g_k ≤ 0;
    return x;
  end;
```

Figure 4. *1-opt* local search.

```
procedure Local-Search-1-opt(x ∈ X): X;
  begin
    calculate gains g_i for all i in {1, ..., n};
    repeat
      find k with g_k = max_i g_i;
      if g_k > 0 then
        x_k := 1 − x_k;
        update gains g_i;
      endif
    until g_k ≤ 0;
    return x;
  end;
```

Figure 5. Fast *1-opt* local search for the BQP.

considerably. The gains g_i do not have to be recalculated each time. Instead, it is sufficient to calculate the difference of the gains Δg_i. Assuming that all g_i for a BQP solution have been calculated and the bit k is flipped, the new gains g_i' can be calculated efficiently by the formula:

$$g_i' = \begin{cases} -g_i & \text{if} \quad i = k \\ g_i + \Delta g_i(k) & \text{otherwise} \end{cases} \quad \text{with} \quad \Delta g_i(k) = 2q_{ik}(\bar{x}_i - x_i)(x_k - \bar{x}_k) \tag{15}$$

Thus, the update of the gains can be performed in linear time. This property has also been used in (Glover et al., 1998b) to speed up tabu search for the BQP. Furthermore, only the gains g_i for $q_{ik} \neq 0$ have to be updated. The fast *1-opt* local search is displayed in figure 5. The running time of this algorithm is $O(n)$ per iteration. The initialization of the gains is in $O(n^2)$.

3.2.2. k-opt local search. The *k-opt* neighborhood $\mathcal{N}_{k\text{-}opt}$ of a binary vector of length n is defined by the binary vectors that can be reached by flipping one up to k bits in the vectors simultaneously. Hence, the neighborhood $\mathcal{N}_{k\text{-}opt}(x) = \{x' \in X \mid d_H(x, x') \leq k\}$ (d_H denotes the hamming distance between bit vectors) grows exponentially with k: $|\mathcal{N}_{k\text{-}opt}| = n^k$.

Since it is computationally too expensive to search the complete *k-opt* neighborhood, Lin and Kernighan have proposed heuristics for the traveling salesman problem (TSP) and the graph partitioning problem that efficiently search a small fraction of the *k-opt* neighborhood. These algorithms, known as the Lin-Kernighan algorithm for the TSP (Lin and Kernighan, 1973), and the Kernighan-Lin algorithm for graph partitioning (Kernighan and Lin, 1972), belong to the best available heuristics for these two combinatorial optimization problems. In the following, a local search algorithm for the BQP is presented that is based on the ideas of Lin and Kernighan.

The basic idea of the heuristic is to find a solution by flipping a variable number of k bits in the solution vector per iteration. In each step, a sequence of n solutions is generated by flipping the bit with the highest associated gain. Analogous to the *1-opt* local search procedure, a vector of gains is maintained and updated according to Eq. (15) after each flip.

```
procedure Local-Search-k-opt(x ∈ X): X;
  begin
    calculate gains g_i for all i in {1,...,n};
    repeat
      x_prev := x, G_max := 0, G := 0, C := {1,...,n};
      repeat
        find j with g_j = max_i g_i;
        G := G + g_j;
        if G > G_max then
          G_max := G;
          x_best := x;
        endif
        x_j := 1 − x_j;
        update gains g_i for all i;
        C := C\{j};
      until C = ∅;
      if G_max > 0 then
        x := x_best;
      else
        x := x_prev;
      endif
    until G_max ≤ 0;
    return x;
  end;
```

Figure 6. *k-opt* local search for the BQP.

Furthermore, a candidate set is used to assure that each bit is flipped exactly once. The best solution in the sequence is accepted as the new solution for the next iteration. Thus, in each iteration of the algorithm a variable number of bits are flipped to find a new solution in the neighborhood of the current solution. The pseudo code of the *k-opt* local search is presented in figure 6.

The runtime complexity for the initialization is in $O(n^2)$ and the running time per iteration is also of complexity $O(n^2)$.

The reduce the running time, the termination condition of the inner repeat-loop can be modified so that the loop is terminated if there were no new x_{best} for more than m iterations. Thus, the resulting *fast k-opt* procedure has a shorter running time for $m \ll n$ than the *k-opt* procedure described before.

4. Performance evaluation

To evaluate the performance of the algorithms, we conducted several experiments on all 105 problem instances contained in ORLIB (Beasley, 1990). The sets glov-a, glov-b, glov-c and glov-d contain the instances of type a through d described in Glover, Kochenberger, and Alidaee (1998a). The set glov200 (glov500) consists of five problem instances of size $n = 200$ ($n = 500$) and is denoted as type e (f) in Glover, Kochenberger, and Alidaee

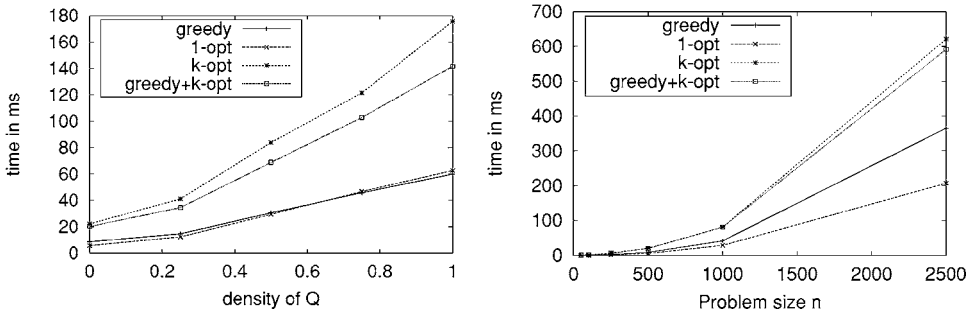

Figure 7. Average running times of greedy and local search heuristics for the BQP.

(1998a). The six sets described in Beasley (1998) with a density $dens(Q)$ of 0.1 and $n \in \{50, 100, 250, 500, 1000, 2500\}$ consist of 10 instances each. They are denoted beas$\langle n \rangle$. In the experiments, the instances of type g used in Glover, Kochenberger, and Alidaee (1998a) and Amini, Alidaee, and Kochenberger (1999) with $n = 1000$ and $dens(Q)$ between 0.1 and 1 were also considered. They are denoted as kb-g.

To find a good parameter value for m in the *k-opt* procedure, experiments were performed for beas1000 and beas2500. It appeared that $m = 100$ is a good trade-off between running time and solution quality. For larger values, the running time increased considerably with only small changes in solution quality. Therefore $m = 100$ was chosen in all subsequent *k-opt* local search experiments.

In the experiments, the performance of the randomized greedy algorithm, the *1-opt* local search applied to randomly generated solutions, the fast *k-opt* local search applied to randomly generated solutions, and the combination of the randomized greedy heuristic and fast *k-opt* local search was investigated. To enable a fair comparison, all algorithms implemented in $C++$ were run under the same conditions on a Pentium II PC (300 MHz) under the operating system Solaris 2.6.

In a first experiment, the average running times of the four algorithms and the average solution quality was studied. In figure 7, the average running times (in milliseconds) of 10000 runs of the algorithms are provided. In the left plot, the running times are provided for the five instances of the set glov500 with $n = 500$ and a density $dens(Q)$ contained between 0.1 and 1. As expected, the running time of the algorithms grows linearly with the density of the matrix Q. The running times of the combination of the randomized greedy heuristic and *k-opt* local search are slightly lower than the running times of the *k-opt* local search applied to randomly generated solutions, since the number of iterations of the local search is reduced when it is applied to solutions produced by the greedy heuristic. In the right plot, the running times are provided for the six sets beas50 to beas2500. For all algorithms, the running times grow quadratically with n. The *k-opt* algorithms appear to be 2.5 times slower than the *1-opt* algorithm, and for large n, the greedy heuristic is slower than *1-opt*. Thus, the number of iterations of the *1-opt* procedure grows at most linearly with the problem size for the instances studied.

The average solution quality of the approaches is displayed in Table 1. The solution quality is measured by the average percentage excess (avg) $(100 \cdot (1.0 - f(x)/f_{\text{best}}))$ over

Table 1. Average solution quality of greedy and local search heuristics for the BQP.

	Greedy		*1-opt*		*k-opt*		Greedy-*k-opt*	
Instances	Avg (%)	Sdev	Avg (%)	Sdev	Avg (%)	Sdev	Avg (%)	Sdev
glov-a	3.83	1.33	2.02	0.83	0.38	0.30	0.20	0.27
glov-b	42.98	8.25	29.44	5.31	14.69	4.54	19.76	6.03
glov-c	3.52	0.60	1.21	0.78	0.24	0.24	0.19	0.14
glov-d	2.81	0.42	2.71	0.73	0.71	0.35	0.42	0.27
glov200	2.41	0.77	1.99	0.96	0.50	0.31	0.31	0.15
glov500	1.84	0.12	1.95	0.36	0.56	0.09	0.31	0.10
beas50	4.31	1.92	5.20	3.57	0.89	0.82	0.55	0.50
beas100	2.37	0.94	3.02	1.54	0.65	0.46	0.49	0.56
beas250	2.09	0.53	2.44	1.12	0.65	0.45	0.41	0.24
beas500	1.73	0.35	2.12	0.48	0.62	0.23	0.48	0.18
beas1000	1.67	0.15	1.71	0.24	0.54	0.12	0.39	0.08
beas2500	1.38	0.13	1.15	0.13	0.40	0.07	0.29	0.07

the best-known solution for a set of up to 10 instances, and the standard deviation (sdev) is also provided for each algorithm. The greedy heuristic shows a better average performance on six of the sets than the *1-opt* local search, while the latter performs better on the remaining six. The *k-opt* heuristic performs considerably better than the greedy and *1-opt* heuristic: with one exception, the average percentage excess is below 1%. However, the combination of greedy and *k-opt* local search performs best on all but one instance with respect to the average quality of the solutions.

In a second experiment, the heuristics were repeatedly applied (multi-start) to show their ability to reach the optimum or best-known solution. Each of the heuristics was started multiple times and the best solution found was returned. For each instance, 30 runs were performed for each algorithm, and the times to reach the best-known solutions were recorded. In Table 2, the times for the algorithms that were capable of finding the optimum in all 30 out of 30 runs for all instances in a set are displayed. The average number of repetitions needed (rep), the average time in seconds (avg t) to reach the best-known solution, as well as the maximum time in seconds (max t) to reach the best-known solution is provided. For problems up to a size of $n = 200$, the *1-opt* local search is capable of finding the optimum in less than 2 seconds. For the set glov200 the average number of local searches is about 217. The *k-opt* heuristic needs only about 13 local searches on the average to find the best-known solutions for the instances of this set. Both *k-opt* algorithms perform better on the instances up to $n = 200$; they need less than 0.23 seconds to find the best-known solutions and are able to find the best-known solutions of all the instances of the set beas250. On this set, the greedy *k-opt* combination performs slightly better than the *k-opt* on random solutions: less than 0.53 seconds are needed.

The results show that for small instances (up to $n = 250$), a simple local search is sufficient to find best-known solutions quickly. The more challenging problems have a size of

Table 2. Time to reach the optimum.

	1-opt			*k-opt*			Greedy-*k-opt*		
Instances	Rep	Avg t	Max t	Rep	Avg t	Max t	Rep	Avg t	Max t
glov-a	8.88	0.01	0.04	2.12	0.01	0.03	9.75	0.01	0.14
glov-b	16.89	0.03	0.20	3.11	0.02	0.08	5.67	0.02	0.13
glov-c	9.29	0.01	0.05	1.57	0.01	0.02	2.29	0.01	0.03
glov-d	52.11	0.05	0.33	4.11	0.02	0.08	4.67	0.02	0.11
glov200	217.20	0.40	1.80	12.80	0.08	0.21	10.60	0.07	0.22
beas50	10.70	0.01	0.03	1.70	0.01	0.02	3.00	0.01	0.13
beas100	173.40	0.06	0.81	5.40	0.01	0.05	11.90	0.02	0.15
beas250	–	–	–	20.00	0.09	0.77	10.30	0.05	0.52

$n = 500$ and higher. The third experiment concentrated on these instances. To enable a fair comparison, the four algorithms were repeatedly applied until a predefined time limit was reached. Once again, 30 runs were performed for each instance. The results are summarized in Table 3. For each instance and algorithm, the average number of repetitions (rep), and the average percentage excess (avg) is given. The time limit in seconds (time) used is provided in the last column of the table.

The greedy heuristic shows to be inferior to the *1-opt* local search: The average percentage excess for the set beas1000 and beas2500 is between 0.727% and 1.170% in case of the greedy heuristic; for the *1-opt* local search the results are between 0.216% and 0.543%. The algorithms based on *k-opt* are considerably better. The worst performance lies 0.128% below the optimum for the beas instances. The kb-g instances appear to be harder; within a time limit of 30 seconds, the average solution quality lies between 0.012% and 0.489%. A preference to one of the *k-opt* based algorithms can not be given, since their performance does not differ significantly. Both are able to find the best-known solution for the problems glov500-1 and glov500-2 in all 30 runs, but not for the other problems with $n = 500$ and a density $dens(Q)$ greater 0.25. This indicates that problems with high densities are slightly harder for the *k-opt* local search. However, as the results on the kb-g instances show, the average solution quality is not a simple function of the density: the average percentage access for the problem with density 0.8 (kb-g08) is better than for the problems with density 0.4 and 0.7 (kb-g04 and kb-g07).

In comparison to the tabu search and simulated annealing for the BQP proposed in Beasley (1998), the best found solutions obtained with the greedy heuristic and *k-opt* local search for the 10 problems of size 2500 are in 7 out of 10 cases better than the best found solutions reported in Beasley (1998). For tabu search and simulated annealing, the running times on a Silicon Graphics Indigo workstation (R4000, 100 MHz) are between 12721 and 51873 seconds compared to the 60 seconds for the local search on a Pentium II 300 MHz PC. Thus, the results produced by the *k-opt* local search appear to be superior or at least competitive to the other approaches in solution quality per time. However, a comparison of the methods under the same conditions (computing hardware, operating system, programming language, coding techniques, . . .) is required to support this claim.

Table 3. Comparison of greedy, *1-opt*, *k-opt*, and greedy-*k-opt* on large BQP instances.

	Greedy		*1-opt*		*k-opt*		Greedy-*k-opt*		Time
Instances	Rep	Avg (%)	Rep	Avg (%)	Rep	Avg (%)	Rep	Avg (%)	(sec)
glov500-1	1163	0.472	1476	0.100	7	0.000	7	0.000	10
glov500-2	674	0.489	798	0.059	48	0.000	27	0.000	10
glov500-3	321	0.796	329	0.168	85	0.005	66	0.001	10
glov500-4	211	0.824	208	0.170	60	0.009	41	0.011	10
glov500-5	159	0.862	153	0.248	55	0.003	64	0.002	10
kb-g01	785	1.039	1108	0.581	282	0.013	276	0.012	30
kb-g02	478	1.488	574	0.865	196	0.046	184	0.012	30
kb-g03	349	2.739	389	1.383	88	0.017	137	0.085	30
kb-g04	275	2.262	296	1.148	112	0.244	131	0.245	30
kb-g05	227	1.491	235	0.740	84	0.022	103	0.031	30
kb-g06	193	2.810	200	1.570	72	0.283	87	0.188	30
kb-g07	168	3.190	172	2.012	62	0.414	76	0.489	30
kb-g08	148	2.131	150	1.147	52	0.183	67	0.146	30
kb-g09	133	3.083	134	1.760	48	0.479	60	0.320	30
kb-g10	120	2.959	121	1.788	43	0.438	54	0.377	30
beas1000-1	710	0.766	1004	0.221	96	0.000	246	0.006	30
beas1000-2	717	1.086	1025	0.351	280	0.000	340	0.048	30
beas1000-3	708	0.890	1010	0.239	165	0.006	289	0.013	30
beas1000-4	709	0.813	1002	0.216	333	0.015	337	0.025	30
beas1000-5	715	0.984	1019	0.348	255	0.014	360	0.014	30
beas1000-6	704	0.727	985	0.316	236	0.024	321	0.037	30
beas1000-7	711	0.815	1016	0.300	367	0.026	327	0.033	30
beas1000-8	698	1.044	991	0.397	345	0.051	350	0.049	30
beas1000-9	712	1.029	1011	0.422	317	0.014	152	0.001	30
beas1000-10	719	0.838	1020	0.300	348	0.008	354	0.053	30
beas2500-1	164	1.170	275	0.541	97	0.073	100	0.089	60
beas2500-2	164	0.864	276	0.456	97	0.083	102	0.106	60
beas2500-3	164	1.114	274	0.543	97	0.053	100	0.059	60
beas2500-4	164	0.829	275	0.390	100	0.022	93	0.010	60
beas2500-5	164	1.051	273	0.401	98	0.028	101	0.036	60
beas2500-6	165	0.956	275	0.383	101	0.091	101	0.071	60
beas2500-7	164	1.088	275	0.527	98	0.128	99	0.108	60
beas2500-8	164	0.749	275	0.289	102	0.073	98	0.024	60
beas2500-9	165	0.985	273	0.359	98	0.063	101	0.057	60
beas2500-10	165	0.941	276	0.450	95	0.110	101	0.080	60

More sophisticated algorithms, however, are expected to produce better solutions: the tabu search utilizing critical event memory (Glover et al., 1998b) found the best-known solution for 7 out of the 10 instances of set kb-g, in a CPU time of four minutes on a Pentium 200 PC. The scatter search approach proposed in Amini, Alidaee, and Kochenberger (1999) found the best-known solutions of all problems in this set, but no CPU times were reported. The heuristics proposed in this paper are not intended to be competitive with these approaches. Instead, they have been developed with the idea of building powerful meta-heuristics called memetic algorithms (Moscato, 1999; Merz and Freisleben, 1999b)—evolutionary algorithms incoporating greedy heuristics and local search.

5. Conclusions

In this paper, a randomized greedy heuristic and two local search algorithms based on the *1-opt* and a *k-opt* neighborhood for the BQP have been proposed. The runtime behavior and the average performance of these heuristics has been investigated in experiments on 115 problem instances. Furthermore, four algorithms based on these heuristics have been studied: the multi-start randomized greedy algorithm, the multi-start *1-opt* local search on randomly generated solutions, the multi-start *k-opt* local search on randomly generated solutions, and the multi-start *k-opt* local search on solutions generated by the randomized greedy heuristic. The results indicate that multi-start *1-opt* local search is able to find best-known solutions of the studied instances up to a size of $n = 100$ in all runs, and multi-start *k-opt* local search is capable of finding the best-known solutions in all runs for all studied instances up to $n = 250$. Furthermore, for 11 out of 15 problem instances of size $n = 500$, the best-known solution was found in all runs with multi-start *k-opt* LS.

Applied to larger instances ($n > 500$), the multi-start randomized greedy heuristic produces solutions less than 1% below the best-known solutions in most cases, the multi-start *1-opt* local search solutions below 0.5% in most cases and the multi-start *k-opt* algorithms solutions less than 0.5% below the best-known solution in most cases in a CPU time limit of 10 seconds up to a minute on a state-of-the-art workstation. Thus, the proposed *k-opt* local search appears to be highly effective, and the additional coding effort required in comparison to the *1-opt* heuristic is definitely justified.

Both greedy and local search heuristics are well suited as components for meta-heuristics since their average running time is below 700 milliseconds even for large problems of size $n = 2500$. If best-known solutions are desired for large problems, the incorporation of the heuristics in genetic algorithms appears to be a promising approach (Merz and Freisleben, 1999a). Further studies have shown that the combination of evolutionary algorithms and *k-opt* local are highly effective applied to large problem instances ($n \geq 500$) (Katayama and Narihisa, 2000; Merz, 2000).

There are several areas of future research. First, the implementation specific parameter m of the *k-opt* local search should be studied in more detail; an interesting question is whether there is an instance independent optimum value for the parameter or whether the parameter has to be tuned for each instance separately. Second, there may be instances for which the greedy heuristic performs better than local search, as shown for the graph bipartitioning problem (Merz and Freisleben, 2000). Additional experiments are necessary to support

this claim. Third, a comparison of the *k-opt* local search with the recently proposed tabu search algorithms for the BQP (Beasley, 1998) is another interesting are of study. Finally, the heuristics described in this paper may be extended to NK landscapes, a generalized unconstrained binary programming problem. In fact, we are currently working on a greedy and a *k-opt* heuristic for NK landscapes. Initial experiments indicate that both heuristics are highly efficient.

References

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

Amini, M.M., B. Alidaee, and G.A. Kochenberger. (1999). "A Scatter Search Approach to Unconstrained Quadratic Binary Programs." In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. London: McGraw-Hill, pp. 317–329.

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

Battiti, R. and A. Bertossi. (1998). "Differential Greedy for the 0–1 Equicut Problem." In D. Du and P. Pardalos (eds.), *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilities Location*, Vol. 40 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Providence, RI: American Mathematical Society, pp. 3–21.

Beasley, J.E. (1990). "OR-Library: Distributing Test Problems by Electronic Mail." *Journal of the Operational Research Society* 41(11), 1069–1072.

Beasley, J.E. (1998). "Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem." Technical Report, Management School, Imperial College, London, UK.

Billionnet, A. and A. Sutter. (1994). "Minimization of a Quadratic Pseudo-Boolean Function." *European Journal of Operational Research* 78, 106–115.

Corne, D., M. Dorigo, and F. Glover (eds.) (1999). *New Ideas in Optimization*. London: McGraw-Hill.

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

Garey, M.R. and D.S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman.

Glover, F., G.A. Kochenberger, and B. Alidaee. (1998a). "Adaptive Memory Tabu Search for Binary Quadratic Programs." *Management Science* 44(3), 336–345.

Glover, F., G. Kochenberger, B. Alidaee, and M. Amini. (1998b). "Tabu Search with Critical Event Memory: An Enhanced Application for Binary Quadratic Programs." In S. Voss, S. Martello, I. Osman, and C. Roucairol (eds.), *Meta-Heuristics—Advances and Trends in Local Search Paradigms for Optimization*. Dordrecht: Kluwer Academic Publishers, pp. 83–109.

Helmberg, C. and F. Rendl. (1998). "Solving Quadratic (0,1)-Problems by Semidefinite Programs and Cutting Planes." *Mathematical Programming* 82, 291–315.

Ivănescu, P.L. (1965). "Some Network Flow Problems Solved with Pseudo-Boolean Programming." *Operations Research* 13, 388–399.

Katayama, K. and H. Narihisa. (1999). "Performance of Simulated Annealing-Based Heuristic for the Unconstrained Binary Quadratic Programming Problem." Technical Report, Dept. of Information and Computer Engineering, Okayama University of Science, Okayama, Japan.

Katayama, K. and H. Narihisa. (2000). "Solving Large Binary Quadratic Programming Problems by Effective Genetic Local Search Algorithm." In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*. San Mateo, CA: Morgan Kaufman, pp. 643–650.

Kauffman, S.A. and S. Levin. (1987). "Towards a General Theory of Adaptive Walks on Rugged Landscapes." *Journal of Theoretical Biology* 128, 11–45.

Kernighan, B. and S. Lin. (1972). "An Efficient Heuristic Procedure for Partitioning Graphs." *Bell Systems Journal* 49, 291–307.

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

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

Lin, S. and B. Kernighan. (1973). "An Effective Heuristic Algorithm for the Traveling Salesman Problem." *Operations Research* 21, 498–516.

Lodi, A., K. Allemand, and T.M. Liebling. (1999). "An Evolutionary Heuristic for Quadratic 0–1 Programming." *European Journal of Operational Research* 119, 662–670.

McBride, R.D. and J.S. Yormark. (1980). "An Implicit Enumeration Algorithm for Quadratic Integer Programming." *Management Science* 26(3), 282–296.

Merz, P. (2000). "Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies." Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany.

Merz, P. and B. Freisleben. (1999a). "Genetic Algorithms for Binary Quadratic Programming." In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith (eds.), *GECCO-1999: Proceedings of the Genetic and Evolutionary Computation Conference*. San Mateo, CA: Morgan Kaufman, pp. 417–424.

Merz, P. and B. Freisleben. (1999b). "Fitness Landscapes and Memetic Algorithm Design." In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. London: McGraw-Hill, pp. 245–260.

Merz, P. and B. Freisleben. (2000). "Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning." *Evolutionary Computation* 8(1), 61–91.

Moscato, P. (1989). "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms." Technical Report No. 790, Caltech Concurrent Computation Program, California Institue of Technology.

Moscato, P. (1999). "Memetic Algorithms: A Short Introduction." In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. London: McGraw-Hill, pp. 219–234.

Pardalos, P.M. and G.P. Rodgers. (1990). "Computational Aspects of a Branch and Bound Algorithm for Unconstrained Quadratic Zero-One Programming." *Computing* 45, 131–144.

Pardalos, P.M. and G.P. Rodgers. (1992). "A Branch and Bound Algorithm for the Maximum Clique Problem." *Computers and Operations Research* 19(5), 363–375.

Pardalos, P.M. and J. Xue. (1994). "The Maximum Clique Problem." *Journal of Global Optimization* 4, 301–328.

Phillips, A.T. and J.B. Rosen. (1994). "A Quadratic Assignment Formulation for the Molecular Conformation Problem." *Journal of Global Optimization* 4, 229–241.