Genetic Algorithms for 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
E-Mail: {pmerz,freisleb}@informatik.uni-siegen.de

Abstract

In this paper, genetic algorithms for the unconstrained binary quadratic programming problem (BQP) are presented. It is shown that for small problems a simple genetic algorithm with uniform crossover is sufficient to find optimum or best-known solutions in short time, while for problems with a high number of variables ($n \geq 200$) it is essential to incorporate local search to arrive at high-quality solutions. A hybrid genetic algorithm incorporating local search is tested on 40 problem instances of sizes containing between $n = 200$ and $n = 2500$. The results of the computer experiments show that the approach is comparable to alternative heuristics such as tabu search for small instances and superior to tabu search and simulated annealing for large instances. New best solutions could be found for 14 large problem instances.

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

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

is maximized. This problem is also known as the *(unconstrained) quadratic bivalent programming problem*, *(unconstrained) quadratic zero–one programming problem*, or *(unconstrained) quadratic (pseudo-) Boolean programming problem* [4].

The BQP is known to be $\mathcal{NP}$-hard [9] and has a large number of applications, for example in capital budgeting and financial analysis problems [16, 18], CAD problems [15], traffic message management problems [8], machine scheduling [1], and molecular conformation [32]. 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 [13, 30, 31].

Several exact methods have been developed to solve the BQP [29, 2, 5, 11], but due to the computational complexity of the problem, heuristics have recently been proposed to find solutions to large problem instances, including *tabu search* [4, 10], *simulated annealing* [4] and *evolutionary algorithms* [17].

In this paper, genetic algorithms based on either crossover or mutation are applied to the BQP, and a hybrid genetic algorithm incorporating a simple local search is presented to solve the BQP. While the genetic algorithms are able to find optimum/best known solutions for small problem instances, the hybrid algorithm is required to find near optimum solutions for large instances. In particular, it is shown that the hybrid algorithm is able to outperform two recent alternative approaches to the BQP, namely tabu search and simulated annealing. To the best of our knowledge, the hybrid algorithm described in this paper represents the first genetic approach applied to large instances of the BQP.

The paper is organized as follows. In section 2, the genetic algorithms for the BQP are described in detail, including the hybrid genetic algorithm. 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 GENETIC ALGORITHMS FOR THE BQP

In the following, two types of genetic algorithms for the BQP are described. The first type consists of simple genetic algorithms with crossover or mutation, and the second type consists of hybrid genetic algorithms incorporating local search.

2.1 GENETIC ALGORITHMS

The application of genetic algorithms [12] to the BQP is straightforward. The individual components used in our approach are described in the following.

Fitness function: The objective function of the BQP (equation (1)) can be used as the fitness function for a genetic algorithm.

Representation: A solution x to the problem can be represented as a binary string x of length n. All bit combinations are feasible and thus no penalty function is needed.

Crossover: Due to the binary representation, classical genetic operators such as single–point, two–point and uniform crossover can be applied without modification to the BQP. The crossover operator used in our experiments is the uniform crossover operator [34].

Mutation: In our algorithms, we use a simple bit–flip mutation operator that flips a single bit of the bit string constituting a solution to the BQP.

Selection: There are many different selection techniques for genetic algorithms which can be divided into selection–for–reproduction and selection–for–survival strategies. We use a random selection scheme for selection–for–reproduction. Selection–for–survival is performed by selecting the best out of the pool of parents and newly created offspring, as done, for example, in the $(\mu + \lambda)$ evolution strategy [33]. Additionally, we ensure that every solution is contained only once in the population (duplicate checking).

Diversification/restarts: In our approach, a diversification strategy is borrowed from the CHC algorithm proposed by Eshelman [6]. If the algorithm is said to have converged (the average hamming–distance of the population has dropped below a threshold d=10 or there was no change in the population for more than 30 generations), the whole population is mutated except the best individual and the search is restarted. This kind of selection/diversification strategy has been shown to be very effective for small populations and has been used in our algorithms to solve the TSP [20, 25], QAP [19, 23], GBP[21, 26] and NK-landscapes [22].

2.2 HYBRID GENETIC ALGORITHMS

A promising approach to use domain knowledge in genetic algorithms is the incorporation of local search. The resulting algorithms, often called genetic local search algorithms (GLS) [35, 7, 36] or memetic algorithms [27, 28, 25], have proven to be highly effective since they combine the advantages of efficient local search for exploiting the neighborhood of a single solution and population based algorithms that effectively explore the search space.

In the GLS approach, all individuals in the population represent local optima. To achieve this, a local search is applied after each application of the genetic operators to create new offspring. To start with a population of local optima, even the initially created solutions are improved by local search. The general outline of the algorithm is shown in Fig. 1. Except for the crossover and the local search described below, the algorithm's components are identical to those described in section 2.1.

In our GLS algorithm for the BQP, we use the 1-opt local search algorithm as described below and a variant of the uniform crossover for recombination. This variant, called HUX, is biased to create offspring that are equally far away from their parents in terms of the hamming distance, similar to the HUX operator described in [6]. The local search applied to the resulting offspring after recombination is restricted to a region of the search space defined by the two parents: the genes with equal values in the two parents are not modified during local search.

Local Search for the BQP

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 one to the current solution. In our implementation, we search for the solution with the highest fitness, i.e. we search for a flip with the highest associated gain in fitness ($g = f_{new} - f_{old}$). The gain g_k of

```
procedure GLS;
  begin
    initialize population P;
    foreach individual i ∈ P do i := Local-Search(i);
    repeat
      for i := 1 to #crossovers do
        select two parents i_a, i_b ∈ P randomly;
        i_c := Crossover(i_a, i_b);
        i_c := Local-Search(i_c);
        add individual i_c to P;
      endfor;
      P := select(P);
      if P converged then
        foreach individual i ∈ P\{best} do i := Local-Search(Mutate(i));
      endif
    until terminate=true;
  end;
```

Figure 1: The GLS Algorithm

flipping bit k in the current solution can be calculated in linear time using the formula

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

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

The local search algorithm is given in pseudo code in Fig. 2.

```
procedure Local-Search(x ∈ {0,1}^n): {0,1}^n;
  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 2: Local Search

In our implementation, we use a particular data structure for maintaining the gains g_i to speed up the computation. A detailed description can be found in [24].

2.3 EPISTASIS IN THE BQP

There is a close relation between binary quadratic programming and NK-landscapes as defined by Kauffman [14]. 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 site i in the genome:

$$f(x) = \sum_{i=1}^{n} f_i(x_i, x_{i_1}, \ldots, x_{i_{k(i)}}), \quad (3)$$

$$f_i(x) = \sum_{j=1}^{n} q_{ij}\, x_i\, x_j \quad (4)$$

Similar to the NK-landscapes defined in [14], 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. Hence, the degree of epistasis in a BQP instance can be easily determined by calculating the density of the matrix Q. It is defined as the number of non-zero entries divided by the number of total entries in the matrix. Thus, the density is between 0 and 1, where 0 means no epistasis and 1 maximum epistasis (every gene depends on the values of all other genes).

The BQP can be viewed as a special class of NK-landscapes with a special fitness contribution function f_i. Therefore, it is expected that genetic algorithms work well for BQP instances with low epistasis, since GAs have been shown to perform well for NK-landscapes with low epistasis [22]. The experiments presented in the next section were performed

Table 1: Comparison of mutation vs. crossover based algorithm

				Uniform Crossover			Mutation		
instance	n	dens	best	avg.	N_{opt}	t/s	avg.	N_{opt}	t/s
glov-3a	70	0.11	6037	6036.1 (0.014%)	17/30	38	6036.1 (0.014%)	17/30	30
glov-5a	50	0.21	5737	5734.5 (0.044%)	27/30	12	5735.3 (0.029%)	28/30	6
glov-8b	90	1.00	145	142.9 (1.471%)	22/30	23	145.0 (0.000%)	30/30	1
glov-10b	125	1.00	154	148.1 (3.831%)	20/30	29	154.0 (0.000%)	30/30	5
glov-1d	100	0.11	6333	6293.0 (0.632%)	15/30	33	6324.5 (0.134%)	27/30	8
glov-2d	100	0.22	6579	6543.2 (0.544%)	22/30	26	6559.5 (0.297%)	25/30	14
glov-3d	100	0.30	9261	9217.8 (0.466%)	8/30	44	9238.6 (0.242%)	16/30	28
glov-4d	100	0.41	10727	10713.6 (0.125%)	24/30	23	10718.1 (0.083%)	26/30	8
glov-5d	100	0.50	11626	11599.6 (0.227%)	12/30	44	11608.7 (0.148%)	16/30	29
glov-6d	100	0.61	14207	14163.8 (0.304%)	12/30	37	14172.2 (0.245%)	12/30	36
glov-7d	100	0.70	14476	14433.2 (0.295%)	19/30	28	14466.6 (0.065%)	28/30	9
glov-8d	100	0.80	16352	16317.8 (0.209%)	21/30	21	16352.0 (0.000%)	30/30	5
glov-9d	100	0.89	15656	15610.8 (0.289%)	9/30	42	15590.0 (0.422%)	13/30	34
glov-10d	100	1.00	19102	19064.0 (0.199%)	18/30	32	19102.0 (0.000%)	30/30	2

with several instances of low, medium, and high epistasis.

3 COMPUTER EXPERIMENTS

We conducted several experiments by applying our algorithms to 75 different instances contained in ORLIB [3] – 35 small instances ($n \leq 125$) [10], 10 instances of medium size ($n = 200, n = 500$) [10], and 30 large instances with n up to 2500 [4]. The population size was set to $P = 100$ in case of the genetic algorithms without local search, and in the presence of local search a population size of $P = 40$ was used. For all experiments, the recombination/mutation rate was set to 0.5, i.e. $0.5 \cdot P$ offsprings were created in each generation. During the restarts, the individuals were mutated by flipping a third of all the bits in the bit vector.

The genetic algorithms described above, including the local search heuristic, were implemented in C++. All experiments were performed on a Pentium II PC (300 MHz) under the operating system Solaris 2.6. The algorithms were terminated when the best/known solution was found or a predefined time limit was reached.

In our first experiment, we applied the genetic algorithm with uniform crossover as well as our genetic algorithm solely based on mutation to the set of 35 small instances. The time limit was set to 60 seconds per run. The results are shown in Table 1. In the first four columns, the name of the instance, the problem size n, the density (dens) of matrix Q, and the fitness of the optimum or best known solution (best), respectively, is provided. The fifth and the sixth columns contain the results for two genetic algorithms, the first using uniform crossover and the second using the mutation operator described above. These columns contain the average final fitness values (avg.) out of 30 runs, the number of runs in which the optimum/best solution could be found (N_{opt}), and the average runtime of the algorithm in seconds (t/s). Only those instances are listed for which the best known solution could not be found in *all* the runs. For the remaining 21 instances, the optimum/best known solution could be found in 1 – 8 seconds.

The results in Table 1 show that mutation based search is superior to crossover based search for all instances. Uniform crossover yields better average fitness than mutation in case of `glov-9d`, but the mutation GA finds the best known solution more often. The amount of epistasis in the problem expressed by the density of the matrix Q does not seem to influence the "hardness" of an instance since the algorithms do not perform significantly better on instances with low epistasis: other characteristics of the problems seem to have a higher influence. The hardest problems are the problems of type d (`glov-1d` – `glov-10d`) and the corresponding densities have a wide range (from 0.11 up to 1.0).

To investigate the scalability of the algorithms, we performed a second experiment for 20 medium sized instances with varying densities. The time limit was set to 60 seconds for the problems of size $n = 200$, and to 120 seconds for the problems of size $n = 500$. The results for the GA with uniform crossover, the GA with mutation, and the GLS are shown in Table 2.

Again, the density of matrix Q and the best known

Table 2: Comparison of genetic algorithms and genetic local search

			Uniform Crossover			**Mutation**			**GLS**		
instance	dens	best	avg. p	N_{opt}	t/s	avg. p	N_{opt}	t/s	avg. p	N_{opt}	t/s
glov200-1	0.1	16464	0.154	15/30	38	0.117	15/30	32	0.000	30/30	<1
glov200-2	0.2	23395	0.156	2/30	56	0.106	14/30	36	0.000	30/30	<1
glov200-3	0.3	25243	1.036	11/30	45	0.980	11/30	41	0.000	30/30	<1
glov200-4	0.4	35594	0.240	5/30	51	0.075	12/30	37	0.000	30/30	<1
glov200-5	0.5	35154	0.610	3/30	55	0.526	10/30	42	0.000	30/30	<1
glov500-1	0.1	61194	0.689	0/30	120	0.359	2/30	115	0.000	30/30	3
glov500-2	0.25	100161	0.802	0/30	120	0.361	0/30	120	0.000	30/30	5
glov500-3	0.5	138035	1.369	0/30	120	0.509	1/30	116	0.000	30/30	32
glov500-4	0.75	172771	1.291	0/30	120	0.446	2/30	117	0.077	0/30	120
glov500-5	1.0	190507	2.500	0/30	120	0.775	0/30	120	0.002	9/30	103

solutions are provided for each instance in the second and third columns. For each algorithm, the average deviation from the optimum (avg. p) is given in percent ($p(x) = 100 \cdot (f_{opt}/f(x) - 1.0)$), the number of times ($N_{opt}$) the optimum/best known solution was found, and the average running time in seconds (t/s) is provided. Again, mutation based search outperforms crossover based search, but the hybrid genetic algorithm incorporating local search (GLS) proves to be much more effective. For the instances of size $n = 200$, the running times to reach the best known solution are below one second on the average. Problem glov500-4 appears to be hard for the GLS algorithm: in the time limit of 120 seconds, the best known solution could not be found. However, allowing longer running times (2000 seconds), our algorithm was able to find the best known solution, too (in 3 out of 30 times). Problem glov500-5 also seems to be a harder problem, although in comparison to glov500-4 the best known solution could be found in 9 out of 30 times.

In order to put our results into perspective, we applied the genetic local search algorithm to larger problems of size 500, 1000, and 2500. For these problems, the performance of two alternative approaches, tabu search and simulated annealing, has been reported in [4]. Table 3 displays the results of the genetic local search algorithm on 30 large problems (all with a density of 0.1) in comparison to the results reported in [4].

For the GLS, the average number of generations (gen), the best solution value found (best) and the average final solution over 30 runs (avg.), as well as the runtime in seconds (t/s) are shown. For the tabu search (TS) and simulated annealing (SA) algorithms, the best solution found and the total running times on a Silicon Graphics Indigo (R4000 CPU with 100 MHz) are provided. The figures demonstrate the effectiveness of the proposed GLS approach: for 14 out of the 30 problems, new best solutions could be found, and only for 4 problems one of the competitors found a slightly better solution. However, the comparison should be treated with care since the running times are not directly comparable due to the different hardware/software platforms used.

Another evolutionary approach by Lodi et al. is reported in [17]. The approach is similar to our GLS approach in that all individuals in the population represent local optima. Recombination is performed by utilizing the MinRange algorithm [29] instead of our simple form of crossover. It is unclear how much the heuristic crossover of the authors contributes to the performance of their algorithm. A direct comparison is not possible since they used a different platform and do not provide the average final solution values. For example, the (shortest) running time to reach the best known solution of instance glov500-1 is reported to be 43.06 seconds on a Silicon Graphics INDY R10000sc with 195 MHz. Our algorithm took 3 seconds *on the average* on a Pentium II PC with 300 MHz to find the same solution value.

4 CONCLUSIONS

Genetic algorithms and genetic local search algorithms were presented for solving the binary quadratic programming problem. It was shown that the genetic algorithms are able to find optimum/best known solutions for problems up to a problem size of 200 in short time. For larger problems, the genetic local search algorithms, i.e. genetic algorithms incorporating local search, have been shown to be more effective. They are able to outperform two alternative approaches, tabu search and simulated annealing, on large problem instances. In particular, for 14 problem instances con-

Table 3: Comparison of three algorithms for the BQP

	GLS				TS		SA	
instance	gen	best	avg.	t/s	best	t/s	best	t/s
beas500-1	4361	**116586**	116586.0	120	**116586**	956	**116586**	1006
beas500-2	4611	**128339**	128339.0	120	128223	979	128204	1009
beas500-3	4336	**130812**	130812.0	120	**130812**	987	**130812**	1030
beas500-4	4723	**130097**	130095.7	120	**130097**	1003	130077	1061
beas500-5	4511	**125487**	125487.0	120	**125487**	964	125315	1030
beas500-6	4565	**121772**	121770.2	120	121719	966	121719	1028
beas500-7	4378	**122201**	122201.0	120	**122201**	952	**122201**	1014
beas500-8	4725	**123559**	123535.8	120	**123559**	1006	123469	1050
beas500-9	4668	**120798**	120789.8	120	120797	954	**120798**	998
beas500-10	4422	**130619**	130619.0	120	**130619**	971	**130619**	1012
beas1000-1	3824	**371438**	371304.1	600	**371438**	4608	371134	7150
beas1000-2	3834	**354932**	354862.3	600	**354932**	4514	354637	6794
beas1000-3	3829	**371236**	371233.8	600	371073	4518	371226	6943
beas1000-4	3819	**370675**	370506.0	600	370560	4580	370265	7011
beas1000-5	3796	352730	352687.6	600	**352736**	4512	352297	6939
beas1000-6	3687	**359629**	359487.8	600	359452	4444	359313	6749
beas1000-7	3927	**371193**	371084.9	600	370999	4546	370815	6885
beas1000-8	3491	**351994**	351844.6	600	351836	4461	351001	6961
beas1000-9	3910	**349254**	349253.3	600	348732	4488	348309	6626
beas1000-10	4100	351408	351125.6	600	351408	4474	**351415**	6734
beas2500-1	998	**1515306**	1514804.6	1200	1514971	52011	1515011	63080
beas2500-2	992	**1470470**	1469721.0	1200	1468694	51659	1468850	62787
beas2500-3	1019	**1413671**	1412943.0	1200	1410721	49101	1413083	60963
beas2500-4	978	**1507701**	1507674.2	1200	1506242	50642	1506943	63018
beas2500-5	982	**1491796**	1491623.4	1200	**1491796**	51194	1491465	63470
beas2500-6	987	**1468427**	1467918.2	1200	1467700	51669	**1468427**	63310
beas2500-7	993	1478297	1477101.7	1200	1476059	50798	**1478654**	62833
beas2500-8	1012	1483702	1483226.9	1200	**1484199**	49861	1482953	61918
beas2500-9	989	**1482413**	1481622.9	1200	1482306	51873	1481834	62978
beas2500-10	1010	**1482733**	1481899.2	1200	1482354	50981	1482166	62777

tained in ORLIB, new best known solutions could be found.

There are several issues for future research. First, additional heuristics may be incorporated in the genetic algorithm, such as greedy heuristics for creating the initial population or alternative local search algorithms. Second, heuristic recombination operators could be developed to exploit the structure of the problem more effectively. Third, a search space analysis may reveal the differences of the fitness landscapes of the different types of instances and may help in predicting the performance of genetic algorithms and local search. Finally, another open question is whether an algorithm for the BQP can be utilized in order to efficiently solve problems that are special cases of the BQP, such as the maximum clique problem and the maximum independent set problem.

References

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

[2] F. Barahona, M. Jünger, and G. Reinelt, "Experiments in Qadratic 0-1 Pogramming," *Mathematical Programming*, vol. 44, pp. 127–137, 1989.

[3] J. E. Beasley, "OR-Library: Distributing Test Problems by Electronic Mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[4] J. E. Beasley, "Heuristic Algorithms for the Unconstrained Binary Quadratic Programming

Problem," Tech. Rep., Management School, Imperial College, London, UK, 1998.

[5] A. Billionnet and A. Sutter, "Minimization of a Quadratic Pseudo–Boolean Function," *European Journal of Operational Research*, vol. 78, pp. 106–115, 1994.

[6] L. Eshelman, "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," in *Foundations of Genetic Algorithms*, (G. J. E. Rawlings, ed.), pp. 265–283, Morgan Kaufmann, 1991.

[7] B. Freisleben and P. Merz, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, (T. Bäck, H. Kitano, and Z. Michalewicz, eds.), pp. 616–621, IEEE Press, 1996.

[8] G. Gallo, P. L. Hammer, and B. Simeone, "Quadratic Knapsack Problems," *Mathematical Programming*, vol. 12, pp. 132–149, 1980.

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

[10] 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, 1998.

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

[12] J. Holland, *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

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

[14] S. A. Kauffman and S. Levin, "Towards a General Theory of Adaptive Walks on Rugged Landscapes," *Journal of Theoretical Biology*, vol. 128, pp. 11–45, 1987.

[15] J. Krarup and P. M. Pruzan, "Computer–aided Layout Design," *Mathematical Programming Study*, vol. 9, pp. 75–94, 1978.

[16] D. J. Laughunn, "Quadratic Binary Programming," *Operations Research*, vol. 14, pp. 454–461, 1970.

[17] A. Lodi, K. Allemand, and T. M. Liebling, "An Evolutionary Heuristic for Quadratic 0–1 Programming," Tech. Rep., Departement de Mathematiques, Ecole Polytechnique Federale de Lausanne, Switzerland, 1998.

[18] R. D. McBride and J. S. Yormark, "An Implicit Enumeration Algorithm for Quadratic Integer Programming," *Management Science*, vol. 26, no. 3, pp. 282–296, 1980.

[19] P. Merz and B. Freisleben, "A Genetic Local Search Approach to the Quadratic Assignment Problem," in *Proceedings of the 7th International Conference on Genetic Algorithms*, (T. Bäck, ed.), pp. 465–472, Morgan Kaufmann, 1997.

[20] P. Merz and B. Freisleben, "Genetic Local Search for the TSP: New Results," in *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, (T. Bäck, Z. Michalewicz, and X. Yao, eds.), pp. 159–164, IEEE Press, 1997.

[21] P. Merz and B. Freisleben, "Memetic Algorithms and the Fitness Landscape of the Graph Bi-Partitioning Problem," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature - PPSN V*, (A.-E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds.), pp. 765–774, Springer, 1998.

[22] P. Merz and B. Freisleben, "On the Effectiveness of Evolutionary Search in High–Dimensional NK-Landscapes," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, (D. Fogel, ed.), pp. 741–745, IEEE Press, 1998.

[23] P. Merz and B. Freisleben, "A Comparison of Memetic Algorithms, Tabu Search, and Ant Colonies for the Quadratic Assignment Problem," in *1999 Congress on Evolutionary Computation (CEC'99)*, 1999. to appear.

[24] P. Merz and B. Freisleben, "Greedy and Local Search Heuristics for the Unconstrained Binary Quadratic Programming Problem," Tech. Rep. 99-01, University of Siegen, Germany, 1999.

[25] P. Merz and B. Freisleben, "Fitness Landscapes and Memetic Algorithm Design," in *New Ideas in Optimization*, (D. Corne, M. Dorigo, and F. Glover, eds.), McGraw–Hill, 1999, to appear.

[26] P. Merz and B. Freisleben, "Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning," *Evolutionary Computation*, 1999, to appear.

[27] P. Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms," Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[28] P. Moscato and M. G. Norman, "A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems," in *Parallel Computing and Transputer Applications*, (M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, eds.), (Amsterdam), pp. 177–186, IOS Press, 1992.

[29] P. M. Pardalos and G. P. Rodgers, "Computational Aspects of a Branch and Bound Algorithm for Unconstrained Quadratic Zero–One Programming," *Computing*, vol. 45, pp. 131–144, 1990.

[30] 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, 1992.

[31] P. M. Pardalos and J. Xue, "The Maximum Clique Problem," *Journal of Global Optimization*, vol. 4, pp. 301–328, 1994.

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

[33] H. P. Schwefel, *Numerische Optimierung von Computer–Modellen mittels der Evolutionsstrategie*. Vol. 26 of *Interdisciplinary Systems Research*, Basel: Birkhäuser Verlag, 1977.

[34] G. Syswerda, "Uniform Crossover in Genetic Algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2–9, Morgan Kaufman, 1989.

[35] N. L. J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. van Laarhoven, and E. Pesch, "Genetic Local Search Algorithms for the Traveling Salesman Problem," in *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN I*, (H. P. Schwefel and R. Männer, eds.), (Berlin, Germany), pp. 109–116, Springer, 1991.

[36] T. Yamada and R. Nakano, "Scheduling by Genetic Local Search with Multi-Step Crossover," in *Proceedings of the 4th Conference on Parallel Problem Solving from Nature - PPSN IV*, (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), pp. 960–969, Springer, 1996.