Ant-Based Crossover for Permutation Problems

Jürgen Branke, Christiane Barz, and Ivesa Behrens

Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany
branke@aifb.uni-karlsruhe.de

Abstract. Crossover for evolutionary algorithms applied to permutation problems is a difficult and widely discussed topic. In this paper we use ideas from ant colony optimization to design a new permutation crossover operator. One of the advantages of the new crossover operator is the ease to introduce problem specific heuristic knowledge. Empirical tests on a travelling salesperson problem show that the new crossover operator yields excellent results and significantly outperforms evolutionary algorithms with edge recombination operator as well as pure ant colony optimization.

1 Introduction

Crossover for evolutionary algorithms (EAs) applied to permutation problems is notoriously difficult, and many different crossover operators have been suggested in the literature. Ant colony optimization (ACO), however, seems particularly well suited for permutation problems. In this paper, we propose to hybridize these two approaches in a way that performs better than either of the original approaches. In particular, we design a new crossover operator, called ant-based crossover (ABX), which uses ideas from ACO within an EA framework.

In ACO, new solutions are constructed step by step based on a pheromone matrix which contains information about which decisions have been successful in the past. Furthermore, problem specific heuristic knowledge is usually used to influence decisions. In ABX, a temporary pheromone matrix is constructed based on the parents selected for mating. This temporary pheromone matrix is then used to create one or several children in the standard way employed by ACO. This has several interesting implications: First of all, it is now as easy as in ACO to incorporate problem-specific heuristic knowledge. Furthermore, we gain additional flexibility. For example, it is natural to extend ABX to construct children from more than two parents, or to integrate ACO as local optimizer. Finally, the use of a population allows us to explicitly maintain several different good solutions, which is not possible in pure ACO approaches.

While we do not see any reason why the proposed approach should not be successful on a wide range of permutation problems, in this paper we concentrate on the travelling salesperson problem (TSP). We empirically compare our approach with an evolutionary algorithm with edge recombination as well as a pure ACO algorithm.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 754–765, 2003.

The paper is structured as follows: the next section surveys related work and provides a brief overview on recombination operators for permutation problems as well as on ant colony optimization. In Section 3 we introduce the new ant-based crossover operator. The approach is evaluated empirically in Section 4. The paper concludes in Section 5 with a summary and ideas for future work.

2 Related Work

2.1 Permutation Crossover

Crossover for permutation problems is difficult, and has been discussed in the literature for a long time. Generally, a crossover operator should create feasible offspring by combining parental information in a sensible way. What is to be considered sensible also depends on the application at hand. For example, with regard to a TSP, it seems more important to preserve edges from the parents (i.e. direct adjacencies in the permutation), while for a scheduling problem, it is more important to preserve the general precedence relations (cf. [2]).

Standard one-point or multi-point crossover does not work for permutations, as it would generate infeasible offspring. The crossover operators suggested in the literature are numerous and range from simple approaches such as order crossover [5] or partially mapped crossover [8] to more complicated ones such as distance preserving crossover [7], edge assembly crossover [15], inner-over crossover [18], natural crossover [12], or edge recombination crossover [20]. The difficulty of designing a proper permutation crossover even led some researchers to abandon a permutation representation, and to use e.g. random keys encoding [1] instead.

For TSPs, edge recombination crossover (ERX) seems to be a very effective crossover operator as it is able to preserve more than 95% of the parental edges [20]. We will use it later for comparison with ABX and therefore discuss it here in slightly more detail: Starting from a random city, ERX iteratively constructs a tour. In each step, it first considers the (up to 4) cities that are neighbors (i.e. connected) to the current location in either of the two parents. If at least one of those has not been visited so far, it selects the city which has the fewest yet unvisited other cities as neighbors in the parents. Otherwise, a random successor is selected. For details, see [20].

There have also been attempts to incorporate problem-specific knowlege into the crossover operator. For example, Grefenstette [9] and Tang and Leung [17], propose variants of ERX, which, when they have to choose between parental edges, prefer the short ones. Julstrom and Raidl [11] compare several ways for prefering short edges within an ERX framework, for decisions between parental edges as well as for decisions when all parental edges are unadmissible. In effect, the latter approach comes quite close to the the simplest form of ABX proposed here. Despite of this similarity, it still differs in the way the parental information and the heuristic information are combined. Furthermore, it lacks the whole general ACO framework, which allows us to e.g. additionally use ACO as local optimizer.

2.2 Ant Colony Optimization

Standard ACO: ACO is an iterative probabilistic optimization heuristic inspired by the way real ants find short paths between their nest and a food source. The fundamental principle used by ants for communication is stigmergy, i.e. ants use pheromones to mark their trails. A higher pheromone intensity suggests a better path and consequently inclines more ants to take a similar path.

Transferring these ideas to the artificial scenario of a TSP with n cities, an ACO approach works as follows (cf. [3,6]): In every iteration, a number of m (artificial) ants construct one solution each through all the given n cities. Starting at a random city, an ant iteratively selects the next city based on heuristic information as well as pheromone information. The heuristic information, denoted by η_{ij}, represents a priori heuristic knowledge w.r.t. how good it is to go from city i to city j. For TSPs, $\eta_{ij} = 1/d_{ij}$ where d_{ij} is the distance between city i and j. The pheromone values, denoted by τ_{ij}, are dynamically changed by the ACO algorithm and serve as a kind of memory, indicating which choices were good in the past.

When having inserted city i in the previous step, the next city j is chosen probabilistically according to the following probabilities:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in S} \tau_{ih}^{\alpha} \eta_{ih}^{\beta}}, \tag{1}$$

where S is the set of cities that have not been visited yet, and α and β are constants that determine the relative influence of the heuristic and the pheromone values on the ant's decision.

After each of the m ants have constructed a solution, the pheromone information is updated. First, some of the old pheromone is evaporated on all edges according to $\tau_{ij} \mapsto (1-\rho) \cdot \tau_{ij}$, where parameter $\rho \in (0,1)$ specifies the evaporation rate. Afterwards, a fixed amount Δ of additional pheromone is 'deposited' along all tour edges of the best ant in the iteration. Often, the elitist ant (representing the best solution found so far) is also allowed to deposit pheromone along its path. Each of these positive updates has the form $\tau_{ij} \mapsto \tau_{ij} + \Delta$ for all cities i and j connected by an edge of the respective tour. Initially $\tau_{ij} = \tau_0$ for each edge e_{ij}.

Population-Based Ant Colony Optimization (PACO): The population-based ACO (PACO), which has been proposed by Guntsch [10], is a modification of the standard ACO. The main difference is that the pheromone matrix no longer accumulates the information from all the updates over time, but instead only contains information about a small number k of solutions explicitly maintained in a population. Solution construction is performed probabilistically as in the standard ACO described above. The main change is the pheromone update, which is described in more detail in the next paragraph.

In the beginning, the pheromone matrix is initialized with a constant value τ_0, the solution population with a maximal size of k is empty. Then, in each

of the k first iterations, the iteration's best ant is allowed to lay pheromone ($\tau_{ij} \mapsto \tau_{ij} + \Delta$) on all edges of its tour in the pheromone matrix. Futhermore, the tour is added to the solution population. No pheromone evaporates during the first k iterations. In all subsequent iterations $(k+1), (k+2), \ldots$, the best ant updates as before and is added to the solution population. To keep the population size constant, another solution of the population (usually the worst or the oldest) is deleted, and the respective amount of pheromone is subtracted from the elements of the pheromone matrix corresponding to the deleted solution ($\tau_{ij} \mapsto \tau_{ij} - \Delta$). The information of the deleted ant completely disappears in one iteration. Consequently, the pheromone matrix only preserves information about the k ants currently in the solution population. Observe that in PACO, pheromone values never fall below the initial amount of pheromone τ_0 and never exceed $\tau_0 + k\Delta$.

The fact that the pheromone matrix used in PACO represents only a small number of solutions inspired us to design ABX which shall be described in Section 3.

2.3 Hybrids

A couple of authors have suggested to combine the ideas of ACO and EAs in several ways. Bonabeau et al. [4], for example, propose to optimize ACO parameters using an EA, and Miaghikh and Punch [13,14] design a hybrid which uses a pheromone matrix as well as a complete solution as part of each individual's representation. To the authors' knowledge, no one has ever proposed to use an ACO algorithm to replace crossover in the way presented in this paper.

Many approaches combine metaheuristics with local search for best results [7]. But here we are interested in the workings of the specific crossover operator proposed. Since we were afraid that local search might blur the effects of crossover, we decided to concentrate on crossover alone.

3 Ant-Based Crossover

The fundamental idea of ABX is as follows: In each generation of the EA the parents are regarded as a solution population in the sense of a PACO. Their tour information is used to generate temporary pheromone matrices. These temporary pheromone matrices are then used by ants to generate new solutions. The generated set of solutions is the candidate set for the children returned to the EA. This creates a number of design options which are discussed in the following:

Number of parents: In principle, the temporary pheromone matrix can be created from an arbitrary number of parents, ranging from 1 to the population size p. We denote this parameter $\sharp parents$.

Pheromone matrix initialization: It is important how much influence is given to the parents relative to the basic initialization value $\tau_0 = 1/n$. We tested two basic possibilities:

- *Uniform update:* each parent deposits a pheromone value of $1/\sharp parents$ on each of the edges along its tour.
- *Rank-based update:* The amount of pheromone a parent is allowed to deposit depends on its rank within the set of parent individuals. The individual with rank $i (i = 1 \dots \sharp parents)$ is allowed to deposit

$$\Delta_i = \frac{b}{\sharp parents} - \left(\frac{2b-2}{\sharp parents}\right)\left(\frac{i-1}{\sharp parents - 1}\right)$$

  with $b = 1.5$, which results in a linear weighting from best to worst.

In both cases, the total amount of pheromone in each row of the pheromone matrix is equal to 2. Half of it results from the initialization τ_0 and half of it from the parents' updates.

ACO run: Given a temporary pheromone matrix, we have to decide on the number of iterations $\sharp iter$ we would like to run the ACO, and the number of solutions m that are constructed in each iteration. In case we decide to run the ACO for more than one iteration, a pheromone update strategy has to be chosen as well. We used the standard evaporation strategy in combination with an elite ant for pheromone update, the update value was set to $\Delta = 1/\sharp parents$.

Number of children: The general scheme allows us to create any number of children from a single crossover operation, ranging from one to $m \cdot \sharp iter$. The number of children is henceforth denoted $\sharp children$, and the best $\sharp children$ from the $m \cdot \sharp iter$ generated solutions are returned as children.

4 Empirical Evaluation

For empirical evaluation, we proceed as follows: first, we try to find a reasonable set of the basic EA parameter settings. Parameters are tuned independently for an EA with ERX and an EA with ABX. Then, in a second step we will examine the effect of the parameters and design choices specific to the ant-based crossover. Finally, we will compare our approach to the standard algorithms ACO and EA with ERX on different TSP test instances.

4.1 Test Setup

For the initial parameter tuning, we use the eil101 TSP instance from TSPLIB [16] which has an optimal tour length of 629. Our basic EA uses a $(\mu + \lambda)$-reproduction scheme[1] with tournament selection and tournament size of 2. To keep the number of free parameters small, we fix μ to 50 and only vary λ.

[1] λ children are created in every generation, and then compete with the μ individuals from the last generation's population for survival into the next generation

Mutation swaps the subtour between two randomly selected cities. The first city is selected at random, and the second city is selected in its neighborhood. More specifically, if c_1 is the position of the first city in the current tour, the second city's position is determined using a gaussian distribution with expected value of c_1 and standard deviation σ (result modulo n). The mutation operator is called with probability *mutprob*. If an individual is mutated, at least one swap is performed. Additional swaps are performed with probability *repeatSwap*, which results in a geometric distribution of the number of swaps with mean $1/(1 - repeatSwap)$. All children are created by crossover, i.e. crossover probability is equal to 1.0. Specifically for ABX, parameters α and β are fixed to standard values 1 and 5 respectively.

Each algorithm terminates after a fixed number of $50,000$ evaluations. Note that the EA with ERX always generates one child per crossover and performs λ evaluations per generation of the EA, i.e. the EA runs for $50,000/\lambda$ generations. With ABX, each solution generated by an ant counts as one evaluation, i.e. there are $(\lambda/\sharp children)(m \cdot \sharp iter)$ evaluations per generation of the EA, which can be significantly larger than λ. The number of EA generations is reduced accordingly. Recalculating the fitness after mutation is not counted towards the number of evaluations, since this can be done very efficiently in constant time for the given mutation operator. A comparison based on a fixed number of evaluations implicitly assumes that evaluation is much more time consuming than the crossover operation. This is true for many problems but not for a TSP. On the other hand, fixing the runtime makes the result very much dependent on implementation issues. In our experiments with up to 198 cities, the actual runtime differences between the different examined approaches were negligible. We therefore decided to use a fixed number of evaluations as stopping criterion.

In the results reported below, the performance of each parameter set is averaged over 20 runs with different random seeds. T-tests with significance level of 0.99 are used to analyze significance.

4.2 Basic EA Parameters

The basic EA parameters tuned first are the number of offspring per generation λ, the mutation probability *mutprob*, the expected length of the swapped tour σ, and the mutation frequency *repeatSwap*. With regard to ABX, for the test reported here, we use rank-based update of the parents, two parents per crossover, and a single ant producing a single child based on the temporary pheromone matrix ($\sharp children = 1, m = 1, \sharp iter = 1$).

We test all possible combinations of the parameter settings listed in Table 1. The settings that perform best for ERX are $\lambda = 50$, $mutprob = 0.8$, $\sigma = 15$ and $repeatSwap = 0.1$ which yield a solution quality of 691.8.

For the EA with ABX, $\lambda = 1$ performs slightly (but not significantly) better than $\lambda = 24$. Nevertheless, we chose $\lambda = 24$ for further testing, since $\lambda = 1$ restricts the testing of child-parent combinations too much. The effect of the mutation parameters seem to be relatively small. We select the following parameters for future tests: $mutprob = 0.25$, $\sigma = 1$ and $repeatSwap = 0.1$. It is

Table 1. Tested parameter values for reproduction and mutation, settings chosen for future tests are bold.

	ERX	ABX
λ	1, 25, **50**	1, **24**, 50
mutprob	0.25, 0.6, **0.8**, 1.0	0.0, **0.25**, 0.5, 0.75
σ	3, 10, **15**	**1**, 3, 10
repeatSwap	**0.1**, 0.4, 0.5, 0.6	0.0, **0.1**, 0.5

interesting to note that the results without mutation ($mutprob = 0$) are almost as good. The fact that mutation plays a minor role in ant-based crossover is not really surprising, because variation is introduced implicitly as part of crossover by the way ants construct their tours probabilistically.

4.3 Parameters for Ant-Based Crossover

In this section, we analyze the influence of the parameters and design choices specific to ABX. For that purpose, we test all feasible combinations of the parameters specified in Table 2. Evaporation rate ρ is set to 0.1 where needed. Additionally, we test a large number of combinations with $\sharp children = 8$, $\sharp parents = 1$, $\sharp parents = 50$ as well as $\sharp iter = 15$.

Table 2. Tested parameter values for ABX

parameter	values tested
$\sharp parents$	2, 4, 8
parentalUpdate	constant, rank-based
$\sharp children$	1, 2, 24
m	1, 2, 12, 24
$\sharp iter$	1, 2 or 5

Overall, the approach seems to be rather robust with respect to the parameter settings chosen. The following paragraphs outline the main results for the five examined parameters. Results with respect to a specific parameter are averaged over all settings of the other parameters (as long as they existed for all settings of the examined parameter).

Number of parents: Table 3 shows the best tour length over all performed test runs classified according to the number of parents and the parental update strategy. As can be seen, using two or four parents for crossover is better than only one or more than eight. The differences are statistically significant. Looking at the convergence graphs (not shown), it becomes apparent that increasing the number of parents slows down convergence.

Table 3. Test results depending on the number of parents and the parental update

	parental update			
	all		constant	rank-based
$\sharp parents$	mean	std. error	mean	mean
1	639.61	0.2234	639.61	639.61
2	636.38	0.2095	636.36	636.72
4	636.38	0.1820	636.89	636.33
8	637.68	0.2596	637.91	637.50
50	641.27	0.4559	642.60	639.94
all combinations	637.93	0.1736	638.30	637.56

Parental update: Unsurprisingly, rank-based parental update leads to faster convergence than uniform parental update, due to the additional influence of good parents (convergence curves are not shown due to space limitations). As can be seen in Table 3, the difference of the two update strategies w.r.t. the obtained tour length is rather small, but becomes more pronounced in combination with a large number of parents. As has been noted in the previous paragraph, increasing the number of parents slows down convergence. This effect should be counterbalanced to some degree e.g. by using the rank-based parental update.

Number of children per crossover: The 24 children generated per generation of the EA can be produced by calling the ABX once with $\sharp iter \cdot m > 24$. Alternatively, one may call the ABX several times, thereby splitting the total of 24 children to be generated evenly among the ABXs. Our test results suggest that it is significantly better to generate only a few children per crossover and rather call the ABX more than once with a smaller number of children each. In other words, it seems to be important that the children are generated based on the information from different sets of parents. The reason may be that if all 24 children are based on one temporary pheromone matrix, they might be so similar that they lead to early convergence of the EA. Overall, test runs converge slower with decreasing $\sharp children$, but to a better solution (cf. Figure 1). This effect is strengthened with increasing $\sharp iter$ (see below).

Number of ants per iteration: Increasing the number of ants m per ACO iteration implicitly leads to better children. On the other hand, the number of fitness evaluations required per generated child is increased, meaning that the EA can only run for fewer generations. Our tests show that the parameter has little influence on the final results, although convergence is slowed down a bit with increasing m. Apparently, the effect of improved children is not able to outweigh the reduction of EA generations, at least not given the limit of 50,000 evaluations (cf. Table 4). For our test environment, between two and twelve ants per iteration seem to perform best.

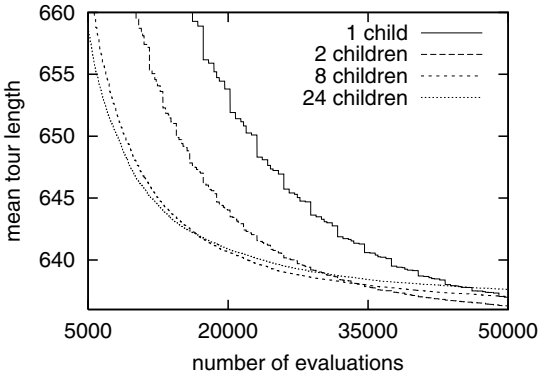

Fig. 1. Convergence behavior of runs with different numbers of children per crossover.

Number of ACO iterations: Similar to increasing the number of ants per iteration, increasing the number of iterations per ACO improves the quality of the generated children at the expense of requiring a larger number of fitness evaluations. Although the additional search should be more structured, when comparing Tables 4 and 5, little difference can be observed regarding the effect of these two parameters. According to our test results, two or five iterations of ants yield the shortest tours. These two settings are significantly better than only a single iteration (cf. Table 5).

Note that the standard error of the results for 15 iterations is relatively high. As can be seen in Figure 2, this high variance can be traced back to two different effects. First of all, in case all children of one generation are generated from a single ACO run, 15 generations lead to premature convergence after only $15,000 - 20,000$ evaluations and very poor results. The effect of many children generated from a single temporary pheromone matrix, as has been described above, is emphasized by running many ACO iterations, since the pheromone matrix converges and thus the children become even more similar. If few children are generated, two cases can be distinguished: If m is large, the number of evaluations per child becomes so high that the runs are far from convergence given the maximum of 50,000 evaluations, and consequently the results are rather poor. On the opposite , the algorithm converges and the results are very good if m

Table 4. Test results depending on the number of ants per ACO iteration

m	mean	std. error
1	636.85	0.3200
2	636.33	0.2619
12	636.14	0.2816
24	637.79	0.4289

Table 5. Test results depending on the number of ACO iterations

$\sharp iter$	mean	std. error
1	637.27	0.2458
2	636.54	0.2292
5	636.62	0.3122
15	637.53	0.6761

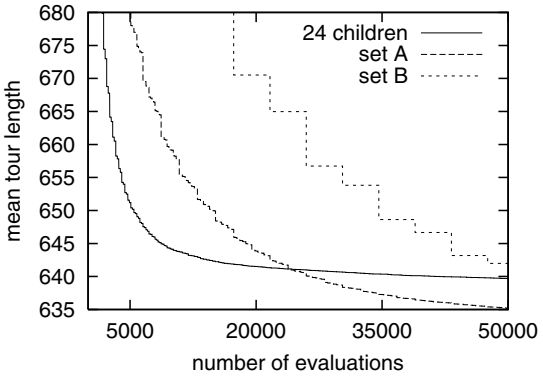

Fig. 2. Convergence behavior of runs with 15 generations of ants. The first line has 24 children per ABX. Sets A and B are averages over runs with $\leq$ 12 children per operator, set A over those with less than 4000 evaluations per crossover, set B over those with more than 4000 evaluations per crossover.

is sufficiently small. On the whole, increasing the number of ACO iterations leads to promising solutions given that the algorithm has sufficient time to converge and the number of children per population is small.

Summary: To sum up, the EA with ABX is quite robust with respect to the examined parameter settings. As is often the case, the ideal parameter settings probably depend on the time available for computation. We have demonstrated that the number of evaluations per crossover operator ($m \cdot \#iter$) plays an important role. If this number is too large, the algorithm will not converge in the given time frame. Apparently, in most cases the effect of local optimization due to the larger number of tours evaluated cannot outweigh the reduction of generations performed by the EA. This stresses the importance of the EA heuristic and clarifies that ABX avails itself of both algorithms and is more than a splitted ACO. For the tests reported in the next section, we use two parents per ABX with uniform update and allow 12 ants to run for 5 iterations to produce one child.

4.4 Comparison of ABX with ERX, and ACO

To compare the performance of our ABX with the other heuristics, we carry out test runs on the following three benchmark problems from the TSPlib [19]: $eil101$ with $50,000$ evaluations, $kroA150$ with $75,000$ evaluations and $d198$ with $100,000$ evaluations (linearly increasing the maximum allowed number of evaluations with the number of cities in the problem). Since in practice, it is not possible to perform extensive parameter tuning when solving a new problem instance, for all heuristics we use the same parameter settings that have proven successful for eil101 respectively. The results are summarized in Table 6.

Table 6. Comparison of the ant-based crossover with other approaches

Heuristic	Problem Instance		
	eil101	kroA150	d198
ERX	691.8	32985.85	18671.8
Standard ACO	638.5	27090.76	16123.36
Ant-Based Crossover	632.5	26807.8	16080.8
Optimum	629	26524	15780

As can be seen, our EA with ABX clearly outperforms the EA with ERX in all tested problem instances. It also performs significantly better than pure ACO[2]. In addition, we can compare ABX to the relatively similar weight-biased edge-crossover reported in [11]. For the tested kroA150 problem, Julstrom and Raidl report an average result of 27081 for their best strategy after 150,000 evaluations, which is clearly inferior to our result of 26807.8 after 75,000 evaluations (at least when ignoring other factors influencing computational complexity).

5 Conclusion and Future Work

In this paper we introduced a new crossover operator for permutation problems which draws on ideas from ant colony optimization (ACO). With the suggested ant-based crossover (ABX), it is straightforward to integrate problem-specific heuristic knowledge and local fine-tuning into the crossover operation. First empirical tests on the TSP have shown that the approach is rather robust with respect to parameter settings, and that it significantly outperforms an EA with edge recombination crossover, as well as pure ACO.

Given these excellent results, the performance of the ABX should also be tested on other permutation problems such as scheduling or the quadratic assignment problem. A more thorough comparison of the computational complexities of the different approaches would also be desirable. Finally, for best results, a hybridization of our approach with local optimizers like Lin-Kernighan should be tested.

References

1. J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2):154–160, 1994.
2. C. Bierwirth, D.C. Mattfeld, and H. Kopfer. On permutation representations for scheduling problems. In H.-M. Voigt, editor, *Parallel Problem Solving from Nature*, volume 1141 of *LNCS*, pages 310–318. Springer, Berlin, 1996.
3. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press, 1999.

[2] $m = 15, \alpha = 1, \beta = 5, \rho = 0.01, \tau_0 = 0.5$, fix update of $\Delta = 0.05$ for best ant of iteration and elite ant, and minimal pheromone value of $\tau_{min} = 0.001$.

4. H. M. Botee and E. Bonabeau. Evolving ant colonies. *Advanced Complex Systems*, 1:149–159, 1998.
5. L. Davis. Applying adaptive algorithms to epistatic domains. In *International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.
6. M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.
7. B. Freisleben and P. Merz. New genetic local search operators for the traveling salesman problem. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature*, volume 1141, pages 890–899, Berlin, 1996. Springer.
8. D. E. Goldberg and R. Lingle. Alleles, loci, and the TSP. In J. J. Grefenstette, editor, *First International Conference on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, 1985.
9. J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann, 1987.
10. M. Guntsch and M. Middendorf. A population based approach for ACO. In *European Workshop on Evolutionary Computation in Combinatorial Optimization*, volume 2279 of *LNCS*, pages 72–81. Springer, 2002.
11. B. A. Julstrom and G. R. Raidl. Weight-biased edge-crossover in evolutionary algorithms for two graph problems. In G. Lamont, J. Carroll, H. Haddad, D. Morton, G. Papadopoulos, R. Sincovec, and A. Yfantis, editors, *16th ACM Symposium on Applied Computing*, pages 321–326. ACM Press, 2001.
12. S. Jung and B.-R. Moon. Toward minimal restriction of genetic encoding and crossovers for the two-dimensional Euclidean TSP. *IEEE Transactions on Evolutionary Computation*, 6(6):557–565, 2002.
13. V. V. Miagkikh and W. F. Punch. An approach to solving combinatorial optimization problems using a population of reinforcement learning agents. In *Genetic and Evolutionary Computation Conference*, pages 1358–1365, 1999.
14. V. V. Miagkikh and W. F. Punch. A generalized approach to handling parameter interdependencies in probabilistic modeling and reinforcement learning optimization algorithms. In *Workshop on Frontiers in Evolutionary Algorithms*, 2000.
15. Y. Nagata and S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In T. Bäck, editor, *International Conference on Genetic Algorithms*, pages 450–457. Morgan Kaufmann, 1997.
16. G. Reinelt. TSPLIB - a travelling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.
17. A.Y.-C. Tang and K.-S. Leung. A modified edge recombination operator for the travelling salesman problem. In *Parallel Problem Solving from Nature II*, volume 866 of *LNCS*, pages 180–188, Berlin, 1994. Springer.
18. G. Tao and Z. Michalewicz. Evolutionary algorithms for the TSP. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 803–812. Springer, 1998.
19. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html.
20. D. Whitley, T. Starkweather, and D'A. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In J. Schaffer, editor, *International Conference on Genetic Algorithms*, pages 133–140. Morgan Kaufmann, 1989.