Iterated Local Search for the Quadratic Assignment Problem

Thomas Stützle
IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, 1050 Bruxelles, Belgium
Email: tstutzle@ulb.ac.be*

Abstract

Iterated local search (ILS) is a surprisingly simple but at the same time powerful metaheuristic for finding high quality approximate solutions for combinatorial optimization problems. ILS is based on the repeated application of a local search algorithm to initial solution which are obtained by mutations of previously found local optima — in most ILS algorithms these mutations are applied to the best found solution since the start of the search.

In this article we present and analyze the application of ILS to the quadratic assignment problem (QAP). We first justify the potential usefulness of an ILS approach to this problem by an analysis of the QAP search space. An investigation of the run-time behavior of the ILS algorithm reveals a stagnation behavior of the algorithm — it may get stuck for many iterations in local optima. To avoid such stagnation situations we propose enhancements of the ILS algorithm based on extended acceptance criteria as well as population-based extensions. A final experimental evaluation of the ILS algorithm and the proposed extensions shows an excellent performance when compared to other state-of-the art heuristic methods for the QAP.

Keywords: Heuristics, Iterated Local Search, Quadratic Assignment Problem, Run-time Distributions, Search Space Analysis

1 Introduction

The quadratic assignment problem (QAP) is an important problem in theory and practice. Many practical problems like backboard wiring [47], campus and hospital layout [11, 13], typewriter keyboard design [7], scheduling [16] and many others [12, 25] can be formulated as QAPs. The QAP can best be described as the problem of assigning a set of facilities to a set of locations with given distances between the locations and given flows between the facilities. The goal then is to place the facilities on locations in such a way that the sum of the product between flows and distances is minimal.

*Currently on leave from FG Intellektik, FB Informatik, TU Darmstadt, Germany. Email: stuetzle@informatik.tu-darmstadt.de

More formally, given n facilities and n locations, two $n \times n$ matrices $A = [a_{ij}]$ and $B = [b_{rs}]$, where a_{ij} is the distance between locations i and j and b_{rs} is the flow between facilities r and s, the QAP can be stated as follows:

$$\min_{\phi \in \Phi} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} a_{\phi_i \phi_j} \tag{1}$$

where Φ is the set of all permutations (corresponding to the assignments) of the set of integers $\{1, \ldots, n\}$, and ϕ_i gives the location of facility i in the current solution $\phi \in \Phi$. $b_{ij} a_{\phi_i \phi_j}$ describes the cost contribution of simultaneously assigning facility i to location ϕ_i and facility j to location ϕ_j. In the following we denote with c_ϕ the solution cost of permutation ϕ.

The QAP is a $\mathcal{NP}$-hard optimization problem; even finding a solution within a factor of $1 + \epsilon$ of the optimal one remains $\mathcal{NP}$-hard [44]. It is considered as one of the hardest optimization problems as general instances of size $n \geq 20$ cannot be solved to optimality. Therefore, to practically solve the QAP one has to apply heuristic algorithms which find very high quality solutions in short computation time. Several such heuristic algorithms have been proposed which include algorithms like iterative improvement, simulated annealing [8, 9], tabu search [3, 45, 52], genetic algorithms [14, 36, 56], evolution strategies [41], GRASP [26], ant algorithms [15, 30, 31, 32, 50, 51], and scatter search [10].

In this article we investigate the performance of iterated local search (ILS) [33, 34, 19, 49] on the QAP. ILS is a very simple and powerful metaheuristic which has proved to be among the best performing approximation algorithms for the well known Traveling Salesman Problem (TSP) [33, 21]. It is based on the observation that iterative improvement algorithms are easily trapped in local minima. Instead of restarting the local search from a new, randomly generated solution, a better idea is often to perturb the current solution (corresponding to a mutation), moving it to a point beyond the neighborhood searched by the local search algorithm. Continuing the local search from such a solution may allow to escape from the current local optimum and to possibly find better solutions. ILS systematically uses this idea by repeatedly applying a local search algorithm to solutions obtained by mutations to one of the previously visited local minima.

In ILS algorithms an acceptance criterion determines to which local optimum the mutation is applied. The acceptance criterion can easily be used to bias the search towards better solutions. In fact, in almost all ILS applications the solution modifications are always applied to the best solution found since the start of the algorithm [21, 33, 38, 19, 18]. This bias in ILS methods is often intuitively justified by the observation that for many problems better solutions have a tendency to be closer to the globally optimal solution [6, 37, 42]. We will present an analysis of the QAP search space which shows that in structured, real life QAP instances such a relation between the distance to an optimal solution and the solution cost exists which intuitively justifies an ILS approach for these problems.

Yet, an investigation of the run-time behavior of a straightforward ILS implementation, which always applies solutions modifications to the best solution found so far, shows that such an ILS algorithm suffers from a type of stagnation behavior. Based on this observation,

we propose to use acceptance criteria which accept occasional moves to worse solutions and in this way provide a higher exploration of possibly better solutions. Additionally, we show that population-based extensions provide a convenient way of avoiding the observed early stagnation phenomena. We present computational results for the proposed ILS algorithms and show that they obtain very high quality solutions, comparable or better than some of the best known heuristic methods for the QAP.

The paper is structured as follows. First, in Section 2, we give a general algorithmic outline of ILS and present in Section 3 the details of the ILS algorithm implementation for the QAP. Section 4 describes the different types of QAP instances used in this paper and in Section 5 we give the results of a search space analysis of the QAP. In Section 6 we present an analysis of the run-time behavior of the ILS approach and based on the results of this analysis we propose some extensions of ILS algorithms. These extensions are then compared in Section 8 to state-of-the-art heuristics for the QAP. We end the article by a summary of the main results, discuss some related work, and present possibilities for future research.

2 Iterated Local Search

Iterated local search (ILS) is a simple and generally applicable metaheuristic that iteratively applies local search to mutations of the current search point. To apply an ILS algorithm to a given problem, four basic components have to be specified. First, an initial solution has to be generated by a procedure GenerateInitialSolution. This initial solution can often be simply a randomly generated solution or one provided by a known construction heuristic. The three main components are a procedure Modify, that mutates the current solution leading to some intermediate solution, a procedure LocalSearch which implements a local search algorithm, and an AcceptanceCriterion that decides to which solution the procedure Modify is applied next. In Figure 1 an algorithmic scheme for ILS is given. Note that, as indicated there, the *history* of the search may be used to influence the solution modification by Modify or the decision which solution is kept to continue the search by AcceptanceCriterion. We will later give some very simple examples of the use of the search history.

ILS is conceptually a rather simple metaheuristic. The simplicity of ILS stems from the underlying principle and the fact that typically only few lines of code have to be added to an already existing local search procedure to implement an ILS algorithm. This is due to the fact that the choice of Modify is often very straightforward and the acceptance criterion is very simple to implement. ILS can also be expected to perform better than to restart local search from randomly generated solution. Thus, it is a rather straightforward way to improve local search performance. Despite its simplicity, it has shown very good computational results for some combinatorial optimization problems like the TSP [34, 33, 21], Graph-Partitioning [33], scheduling problems [29, 48], the p-Median problem [18] and weighted MAX-SAT [19].

It should be noted that in previous research, various terms have been used for algorithms relying on the ILS principle [34, 33, 38]. These algorithms have in common that they rely

```
procedure Iterated Local Search
    s_0 = GenerateInitialSolution
    s = LocalSearch(s_0)
    repeat
        s' = Modify(s, history)
        s'' = LocalSearch(s')
        s = AcceptanceCriterion(s, s'', history)
    until termination condition met
end Iterated Local Search
```

Figure 1: Algorithmic outline of an iterated local search procedure (ILS).

on the same basic principle and they differ mainly in the specific choices for at least one of the three main components of our ILS framework. We use the term iterated local search to generally refer to this type of algorithm. In the following we will shortly discuss some aspects related to the choice of these main components.

In principle, any local search algorithm can be applied, but the performance of the ILS algorithm with respect to solution quality and computation speed strongly depends on this choice. Like in our ILS approach to the QAP, very often a simple iterated descent algorithm is used. Yet, it is also possible to apply more sophisticated local search algorithms like variable depth local search algorithms [27, 21, 33], or short runs of a tabu search or simulated annealing algorithm [28, 29].

The mutation implemented by Modify (we will also refer to it also as a *kick-move* in the following) should be chosen *strong enough* to allow to leave the current local minimum and to enable the local search to find new, possibly better solutions. If the kick-move is too weak the local search algorithm may return after very few steps to the local minimum to which the kick-move has been applied. Often it is sufficient to use for the kick-move a randomly chosen move from some higher order neighborhood than the one used in the local search algorithm. In fact, doing so can be interpreted as a random exploration of higher order neighborhoods. At the same time, the kick-move should be *sufficiently weak* to keep characteristics of the current local minimum since parts of the solution already may be close to optimal. A major problem for too strong kick-moves is that the resulting algorithm would be very similar to repeating local search from randomly generated solutions, which is known to be a weak algorithm for combinatorial optimization problems. Applying only rather small mutations has as an additional advantage that the local search algorithm requires only a few steps to reach the next local optimum, i.e., new local optima can be identified very fast — typically much faster than when starting from a randomly generated solution. Hence, for the same given computation time many more local searches can be performed than when starting from randomly generated solutions. One additional possibility is to choose the strength of the kick-move or its direction based on the search history. A simple mechanism of how to modify the kick-move strength is, for example, introduced in *variable neighborhood search*

(VNS) [38, 19] which can be easily cast into the ILS framework. In VNS the mutations done by Modify are chosen from different neighborhoods $\mathcal{N}_k, k = k_{min}, \ldots, k_{max}$, which are ranked according to some criterion like their size. The larger is the neighborhood the larger is also the strength of the mutation introduced in a solution. In VNS, Modify is applied to the best solution found so far [19], starting from k_{min} and then exploring larger neighborhoods. In case an improved solution is found, the exploration of moves starts again with the first neighborhood $k = k_{min}$. If no better solution is found randomly chosen moves from the next larger neighborhood ($k = k + 1$) are applied to the current solution. Hence, the search history is used to adjust the kick-move strength using a rather simple criterion. This is an important aspect, since it often may be difficult to determine a priori an appropriate kick-move strength, the best kick-move strength may be instance dependent, or even depend on the search space region. Therefore VNS may give more robustness for an ILS algorithm.

The acceptance criterion is used to decide to which solution the next kick-move should be applied. One important aspect of the acceptance criterion is to introduce a bias between exploitation and exploration of the search. Exploitation is achieved, for example, by accepting only better local optima for Modify. Such a choice may be preferable if close to the current local optimum even better solutions may be found (see Section 5 for a more detailed discussion). Exploration of the search space may be achieved, for example, by accepting every new local optimum s''; this would be similar to a random walk over the local optima as the objective function value is not taken into account. Yet, also the search *history* may be used to decide whether some of the earlier found local optima should be chosen or it may influence the decision between s and s''.

3 ILS implementation for the QAP

To apply ILS to the QAP an initial solution has to be generated and the three components LocalSearch, Modify, and AcceptanceCriterion have to be defined. In the following we describe the choices done in our ILS approach to the QAP.

3.1 Choice of LocalSearch

A local search algorithm starts from some initial assignment and repeatedly tries to improve the current assignment by local changes. If in the neighborhood of the current assignment a better assignment ϕ' is found, it replaces the current assignment and the local search is continued from ϕ'.

For the QAP the neighborhood $\mathcal{N}$ of ϕ is defined by the set of permutations which can be obtained by exchanging two facilities at positions r and s, i.e. $\mathcal{N}(\phi) = \{\phi' \mid \phi'_r = \phi_s,\ \phi'_s = \phi_r,\ r \neq s \text{ and } \phi'_i = \phi_i\ \forall i \neq r, s\}$. The objective function difference $\delta(\phi, r, s)$ of exchanging two facilities ϕ_s and ϕ_r can be computed in $O(n)$, using the following equation [54]:

$$\begin{aligned}\delta(\phi, r, s) = a_{rr} \cdot (b_{\phi_s\phi_s} - b_{\phi_r\phi_r}) + a_{rs} \cdot (b_{\phi_s\phi_r} - b_{\phi_r\phi_s}) + \\ a_{sr} \cdot (b_{\phi_r\phi_s} - b_{\phi_s\phi_r}) + a_{ss} \cdot (b_{\phi_r\phi_r} - b_{\phi_s\phi_s}) +\end{aligned}$$

$$\sum_{k=1,k\neq r,s}^{n} (a_{kr} \cdot (b_{\phi_k\phi_s} - b_{\phi_k\phi_r}) + a_{ks} \cdot (b_{\phi_k\phi_r} - b_{\phi_k\phi_s}) + a_{rk} \cdot (b_{\phi_s\phi_k} - b_{\phi_r\phi_k}) + a_{sk} \cdot (b_{\phi_r\phi_k} - b_{\phi_s\phi_k})) \quad (2)$$

Note, that if both matrices A and B are symmetric with a null diagonal, the formula can be simplified using the fact that $a_{ij} = a_{ji}$ and $b_{\phi_i\phi_j} = b_{\phi_j\phi_i}$.

The most simple local search algorithm based on the above described neighborhood is iterative improvement, in the following referred to as `2-opt`. Here we use a first-improvement pivoting rule, i.e., once an improving move is found it is immediately performed. Every full neighborhood scan for the first-improvement heuristic has a complexity of $\mathcal{O}(n^3)$. To improve the speed of the local search algorithm algorithm we adapt a technique called *don't look bits*, which has initially been proposed for local search algorithms applied to the TSP [4, 34]. To apply this technique to local search algorithms for the QAP a so called don't look bit is associated with every item. When first applying local search, all don't look bits are turned off (i.e., they are set to 0). If during the neighborhood scan for an item no improving move is found, the don't look bit is turned on (i.e., set to 1) and the item is not considered as a starting item for a neighborhood scan in the next iteration. Yet, if an item is involved in a move and changes its location, the don't look bit is turned off again.

The don't look bits restrict the attention to the most interesting part of the local search where still further improvement can be expected. Hence, after a solution modification by Modify, which is described in the next subsection, only the don't look bits of items which change their location due to the mutation are reset to 1. Together with this resetting strategy of the don't look bits, the speed of the first-improvement local search is increased considerably (see also Section 8).

3.2 Choice of Modify

To mutate a solution we randomly exchange a number of k items, corresponding to a random move in the k-opt neighborhood. Initially, we tested also other possible kick-moves like exchanging a certain number of pairs or triples of items. We obtained similar results with these other possibilities. Yet, in this article we apply only random k-opt moves since this appears to be a more straightforward choice. To make the particular choice of the kick-move strength, i.e., the value of k, more robust (obviously, a best choice of k is *a priori* not known and appropriate settings for k may depend on the particular search space region), we modify k as done in VNS. If we reach the upper limit $k_{\max}$ without having found an improved solution, we set k to $k_{\min}$ and repeat the same cycle.

3.3 Choice of AcceptanceCriterion

As acceptance criterion in our basic ILS algorithm we use we use $Better(s, s'')$ which is defined as follows:

$$s = Better(s, s'') = \begin{cases} s'' & \text{if } f(s'') < f(s) \\ \\ s & \text{otherwise} \end{cases} \quad (3)$$

Here, $f(s)$ is the objective function value of solution s. This choice of the acceptance criterion appears to be the standard when applying ILS algorithms [35, 33, 21, 19]; only in very few ILS implementations other acceptance criteria are used [34, 28]. For example, in the context of VNS it is argued that exploration of the search space is achieved by allowing larger modifications to the current solution, i.e., by large values for k. Yet, a disadvantage of doing so is that for large k, the modified solution will not differ very strongly from a randomly generated one and the local search will need rather long computation time to find the next local minimum. Again, it is not very clear whether using a high value for k alone is sufficient to allow the algorithm to escape from bad solutions. By performing an analysis of the run-time behavior of ILS applied to the QAP, we will show that this is in fact not the case (see Section 5). Other possibilities like allowing moves to worse solutions by the acceptance criterion have to be used to yield significantly improved performance.

3.4 Initial Solution

As the initial solution we use a randomly generated assignment of items to locations.

4 Types of QAP instances

It is known that the particular type of a QAP instance has a considerable influence on the performance of heuristic methods [54]. According to [54] the instances of QAPLIB which we use in this article can be classified into the following four classes.

(*i*) **Unstructured, randomly generated instances** Instances with the distance and flow matrix entries generated randomly according to a uniform distribution. These instances are among the hardest to solve exactly. Nevertheless most iterative search methods find solutions within $1-2\%$ from the best known solutions relatively fast [54].

(*ii*) **Grid-based distance matrix** In this class of instances the distance matrix stems from a $n_1 \times n_2$ grid and the distances are defined as the Manhattan distance between grid points. These instances have multiple global optima (at least 4 in case $n_1 \neq n_2$ and at least 8 in case $n_1 = n_2$) due to the definition of the distance matrices [54].

(*iii*) **Real-life instances** Instances from this class are "real-life" instances from practical applications of the QAP. Among those are the instances of Steinberg [47] (instances `ste36x`), a layout problem for a hospital [13, 24] (instances `kra30x`), instances corresponding to the layout of typewriter keyboards [7] (instances `bur26x`). Real-life problems have in common that the flow matrices have many zero entries and the remaining entries are clearly not uniformly distributed. The matrix entries exhibit a clear structure and it is in this main aspect that real-life instances differ from the randomly generated instances described in class *i*.

(*iv*) Since the real-life instances in QAPLIB are mainly of a rather small size, a new type of randomly generated problems has been proposed in [54]. These instances are generated in such a way that the matrix entries resemble a distribution found in real-life problems.

To differentiate among the classes of QAP instances the flow dominance *fd* may be used. It is defined as the coefficient of variation of the flow matrix entries multiplied by 100.

$$fd(A) = 100 \cdot \frac{\sigma}{\mu}, \text{ where} \tag{4}$$

$$\mu = \frac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \text{ and } \sigma = \sqrt{\frac{1}{n^2 - 1} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} (a_{ij} - \mu)}$$

A high flow dominance indicates that a large part of the overall flow is exchanged among relatively few items. Hence, randomly generated problems according to a uniform distribution will have a rather low flow dominance whereas real-life problems, in general, have a rather high flow dominance. A disadvantage of the flow dominance is that it captures only the structure of one out of two matrices, neglecting that of the distance matrix. Therefore, analogously to the flow dominance, also a *distance dominance* (*dd*) can be defined.

As indicated before, real life problems often have many zero entries, hence the sparsity of the matrix may also give an indication of the type of an instance. Let n_0 be the number of 0 entries in the flow or distance matrix. Then we define $sp = n_0/n^2$ to be the sparsity of the matrices.

In Table 1 and 2 are given the flow and the distance dominance and the sparsity of the sparser of both matrices (typically, at most one of the two matrices of the QAP instances in Table 1 and 2 has a sparsity larger than 0.1) for some instances of QAPLIB, ordered according to the four classes of instances described above (the other table entries are explained in the next section). Obviously, real-life instances and the randomly generated instances similar to real-life instances have considerably higher dominance values for at least one of the matrices. Also the instances of class *ii* show larger flow dominance than the unstructured, randomly generated instances of class *i*.

5 Analysis of the QAP Search Space

Today it is widely agreed that the performance of metaheuristics depends strongly on the shape of the underlying search space. Central to the search space analysis of combinatorial optimization problems is the notion of *fitness landscape* [46, 58]. Intuitively, the fitness landscape can be imagined as a mountainous region with hills, craters, and valleys. The performance of metaheuristics depends on the ruggedness of the landscape, the distribution of the valleys, craters and the local minima in the search space, and the overall number of the local minima.

Formally, the fitness landscape is defined by

Table 1: Given are the instance identifier, the flow, the distance dominance and the sparsity of some QAPLIB instances (columns 1 to 4). The dominance values are calculated for the first A and the second B matrix as given in QAPLIB. The number in the instance identifier is the instance dimension. The instances are ordered according to the 4 classes described in Section 4. The remaining entries give summary results of an analysis of the fitness-distance correlation of the QAP search space (see Section 5). In particular, N_{opt} is the number of pseudo-optimal solutions found, avg^{ls}_{d-opt} and avg^{ils}_{d-opt} are the average distance to the closest optimum solution for local search and iterated local search executed for n iterations, respectively, and r_{ls} and r_{ils} are the correlation coefficients for the solution cost versus the distance to the closest pseudo-optimal solution.

instance	$dd(A)$	$fd(B)$	sp	N_{opt}	avg^{ls}_{d-opt}	avg^{ils}_{d-opt}	r_{ls}	r_{ils}
unstructured, randomly generated i								
tai20a	67.02	64.90	0.015	1	18.78	18.62	0.065	0.088
tai25a	61.81	64.29	0.016	1	23.83	23.64	0.064	0.059
tai30a	58.00	63.21	0.013	1	28.69	28.32	0.100	0.208
tai35a	61.64	61.57	0.010	1	33.75	33.61	0.041	0.054
tai40a	63.10	60.23	0.009	1	38.86	38.81	0.020	0.107
tai60a	61.41	60.86	0.011	1	58.82	58.71	0.025	0.006
tai80a	59.22	60.38	0.009	1	78.90	78.77	0.022	0.049
rou20	65.65	64.43	0.010	1	18.50	18.10	0.124	0.114
Instances with grid-distances ii								
nug30	52.75	112.48	0.316	4	25.93	23.81	0.262	0.406
tho30	59.25	137.86	0.484	4	26.27	24.86	0.328	0.472
tho40	53.20	155.54	0.585	4	36.11	35.21	0.194	0.273
sko42	51.96	108.48	0.292	4	37.96	35.18	0.302	0.499
sko49	51.55	109.38	0.304	8	44.58	43.40	0.213	0.234
sko56	51.46	110.53	0.305	4	51.62	49.51	0.254	0.448
sko64	51.18	108.38	0.308	8	58.88	56.33	0.303	0.353
sko72	51.14	107.13	0.299	4	67.38	65.31	0.264	0.284

(1) the set of all possible solutions $\mathcal{S}$;

(2) an objective function that assigns to every $s \in \mathcal{S}$ a fitness value $f(s)$;

(3) a distance measure $d(s, s')$ which gives the distance between solutions s and s'.

The fitness landscape determines the shape of the search space as encountered by a local search algorithm.

An ILS algorithm follows a discontinuous trajectorie in such a fitness landscape, embedding an iterative improvement algorithm into a global guiding mechanism which performs jumps in the neighborhood graph used by the iterative improvement method. For such a guiding mechanism to be effective the global characteristics of the fitness landscape topology like the relative location of locally optimal solutions, the average distance between them and the relative location of locally optimal solutions with respect to a globally optimal ones are important. For the investigation of the suitability of a fitness landscape for adaptive multi-start algorithms like the proposed ILS algorithm the analysis of the correlation between solution costs and the distance between solutions has proved to be a useful tool

[6, 22, 23, 40]. In particular two different types of a fitness-distance analysis have been used in the literature.

Table 2: Given are the instance identifier, the flow, the distance dominance and the sparsity of some QAPLIB instances (columns 1 to 4). The dominance values are calculated for the first A and the second B matrix as given in QAPLIB. The number in the instance identifier is the instance dimension. The instances are ordered according to the 4 classes described in Section 4. The remaining entries give summary results of an analysis of the fitness-distance correlation of the QAP search space (see Section 5). In particular, N_{opt} is the number of pseudo-optimal solutions found, avg^{ls}_{d-opt} and avg^{ils}_{d-opt} are the average distance to the closest optimum solution for local search and iterated local search executed for n iterations, respectively, and r_{ls} and r_{ils} are the correlation coefficients for the solution cost versus the distance to the closest pseudo-optimal solution.

instance	$dd(A)$	$fd(B)$	sp	N_{opt}	avg^{ls}_{d-opt}	avg^{ils}_{d-opt}	r_{ls}	r_{ils}
real-life instances *iii*								
bur26a	15.09	274.95	0.223	96	21.12	20.15	0.027	0.457
bur26b	15.91	274.95	0.223	690	21.26	19.72	0.021	0.678
bur26c	15.09	228.40	0.257	96	22.31	15.39	0.569	0.867
bur26d	15.91	228.40	0.257	790	20.29	18.19	0.471	0.787
bur26e	15.09	254.00	0.312	97	18.43	14.91	0.479	0.853
bur26g	15.09	279.89	0.211	96	18.47	13.89	0.666	0.876
chr25a	424.27	57.97	0.883	2	22.92	21.71	0.252	0.359
els19	52.10	531.02	0.637	1	16.85	13.76	0.550	0.654
kra30a	49.22	149.98	0.6	257	25.23	24.10	0.251	0.413
kra30b	49.99	149.98	0.6	128	24.83	23.25	0.312	0.379
ste36a	55.65	400.30	0.707	8	30.98	28.37	0.295	0.504
ste36b	100.79	400.30	0.707	8	29.59	24.76	0.381	0.778
real-life like instances *iv*								
tai20b	128.25	333.23	0.410	1	17.32	14.69	0.420	0.576
tai25b	87.02	310.40	0.387	1	21.65	23.83	0.456	0.703
tai30b	85.20	323.91	0.432	1	27.71	25.47	0.264	0.518
tai35b	78.66	309.62	0.524	1	32.04	30.17	0.328	0.525
tai40b	66.75	317.22	0.503	1	37.76	35.39	0.329	0.626
tai50b	73.44	313.91	0.548	1	47.94	45.39	0.156	0.363
tai60b	76.83	317.82	0.548	1	56.88	52.28	0.366	0.540
tai80b	64.05	323.17	0.552	1	77.47	75.56	0.150	0.457
tai100b	80.42	321.34	0.552	1	95.23	92.34	0.546	0.608

(1) Study the correlation between cost and the average distance to the other local minima [40, 39, 6, 5].

(2) Study the correlation between cost and the distance to the best local minimum or the known global optimum [6, 5, 22, 23, 37, 42].

The fitness distance correlation (FDC) coefficient has been proposed in [22]. It measures the correlation of the solution cost and the distance to the closest global optimum. Given a set of cost values $C = \{c_1, \ldots, c_m\}$ and the corresponding distances $D = \{d_1, \ldots, d_m\}$ to the closest global optimum the correlation coefficient is defined as:

$$r(C,D) = \frac{c_{CD}}{s_C \cdot s_D} \tag{5}$$

where

$$c_{CD} = \frac{1}{m}\sum_{i=1}^{m}(c_i - \bar{c})(d_i - \bar{d}) \tag{6}$$

and $\bar{c}$, $\bar{d}$ are the average cost and the average distance, s_C and s_D are the standard deviations of the costs and distances, respectively. The FDC has shown to be useful to analyze the hardness of a problem for a genetic algorithm, but it is also very useful to give hints on the effectiveness of adaptive algorithms using discontinuous trajectories.

Why is the FDC important for the interpretation of search performance of discontinuous metaheuristics? The task of adaptive restart algorithms which embed local search procedure like iterated local search is to guide the search towards search space regions which contain very high quality solutions and, possibly, the global optimum. The most important guiding mechanism of these methods is the objective function value of solutions. It relies on the general intuition that the better a solution the more likely it is to find even better solutions close to it. On the other side, the notion of *search space region* is tightly coupled to the notion of distance between solutions, as defined by an appropriate distance measure. In particular, the fitness-distance correlation describes the relation between the solution cost and the distance to the globally best solution. For minimization problems a high, positive correlation indicates that the better the solution, the closer — on average — it is to the global optimum. Hence, if such a correlation exists, this gives also a justification of an algorithm like the presented iterated local search algorithm which always applies solution modifications to the current best solution. In case no such correlation exists or it is very weak, the fitness gives only little guidance towards better solutions. Even worse, if cost and distance are negatively correlated, guiding the search by objective function values may be misleading because – on average – the better the solution quality, the farther away the algorithm gets from the global optimum or the best solutions. This last phenomenon has been observed, for example, in deceptive problems for genetic algorithms [22].

Here we perform an analysis of the fitness distance correlation of the QAP search space. As the distance between solutions we use the number of items which are placed on distinct locations in two solutions ϕ and ϕ', i.e., $d(\phi,\phi') = |\{i \mid \phi_i \neq \phi'_i\}|$. To measure the distance to the globally optimal solution we use the optimal ones if they are available. Yet, since exact algorithms generally cannot solve QAP instances with $n \geq 25$ to optimality, we measure the distance to the best known solutions in the fitness distance analysis of instances for which the optimal solution has not yet been proved. These best known solutions are conjectured to be optimal for instances of class *ii*, *iii*, and *iv* up to 80 items since they are the best solutions found by several algorithms (like our ILS algorithms). Since these solutions are not proven to be optimal, we refer to them as pseudo-optimal in the following.

When performing the analysis of the fitness distance correlation of the QAP search space one has to take into consideration the fact that many instances, for example, all instances

with grid-distances, have multiple optimal solutions. These optimal solutions may be at the maximally possible distance from the other optimal solutions, like is the case for the instances of class *ii* (this fact is due to symmetries in the distance matrix). Hence, on instances with multiple pseudo-optimal solutions we measure the distance to the closest global optimum. Since the exact number of global optima for these QAP instances is not known, we determined in preliminary runs of our ILS algorithm a (possibly large) number of pseudo-optimal solutions. For the smaller instance with $n < 40$ we stopped searching for more pseudo-optimal solutions if in 1000 runs of our ILS algorithm, each run of at least 500 iterations, we did not find any more a pseudo-optimal solution with a distance larger than 2 from any previously found pseudo-optimal solution. In case of the larger instances of type *ii* we searched only for the expected number of optimal solutions. On instances of types *i* and *iv* only one single pseudo-optimal solution has been found. Hence, we conjecture that these instances have unique optimal solutions. The number of pseudo-optimal solutions found for each instance are indicated by N_{opt} in Tables 1 and 2.

For the fitness-distance analysis we performed two series of experiments. In a first experiment (denoted as E-ls) we generated 5000 local optima (identical solutions at distance 0 among the generated local optima have been eliminated) with 2-opt and measured the distance to the closest pseudo-optimal solution. In a second series of experiments (E-ils) we run 1000 times our ILS algorithm for n iterations, indicated by ILS(n) (again, identical solutions have been eliminated). This second series of experiments is motivated by the observation that for the TSP the FDC among higher quality solutions is larger and the average distance to the globally optimal solutions is smaller [5]. Here we examine this same aspect for the QAP.

In Tables 1 and 2, where additionally the flow dominance, the distance dominance, and the sparsity are given, we give the number of pseudo-optimal solutions found in the preliminary runs N_{opt}, the average distances of the local minima to the closest global optimum avg^{ls}_{d-opt}, the same for the ILS(n) runs (avg^{ils}_{d-opt}), and finally the empirical FDC coefficients found in the two experiments (r_{ls} and r_{ils}, respectively). Additionally we present in Figure 2 plots of the distance versus the fitness for one instance of each problem class.

The fitness-distance analysis shows clear differences among the behavior for the different problem classes. For class *i*, the correlation coefficients are close to zero for all the instances. Obviously, since these are instances which are randomly generated according to a uniform distribution, one cannot expect to find preferred locations for items. Also the better quality solutions generated by ILS(n) do not show a much higher correlation. Regarding the average distance of the local optima from the global optimum, it can be observed that avg^{ls}_{d-opt} and avg^{ils}_{d-opt} are very large for these QAP instances, close to the maximal possible value which is n. Also for the instances of class *i* the difference between avg^{ls}_{d-opt} and avg^{ils}_{d-opt} is minimal.

For the other three classes, significant correlations between the solution cost and the distance to an pseudo-optimal solution exist. All the correlations are statistically significant at the $\alpha = 0.05$ level with the only exception of r_{ls} for instances `bur26a` and `bur26b`. Yet, the higher quality solutions found by ILS(n) for the two instances again show a highly significant correlation. Differently from instances for class *i*, for the other classes the correlations

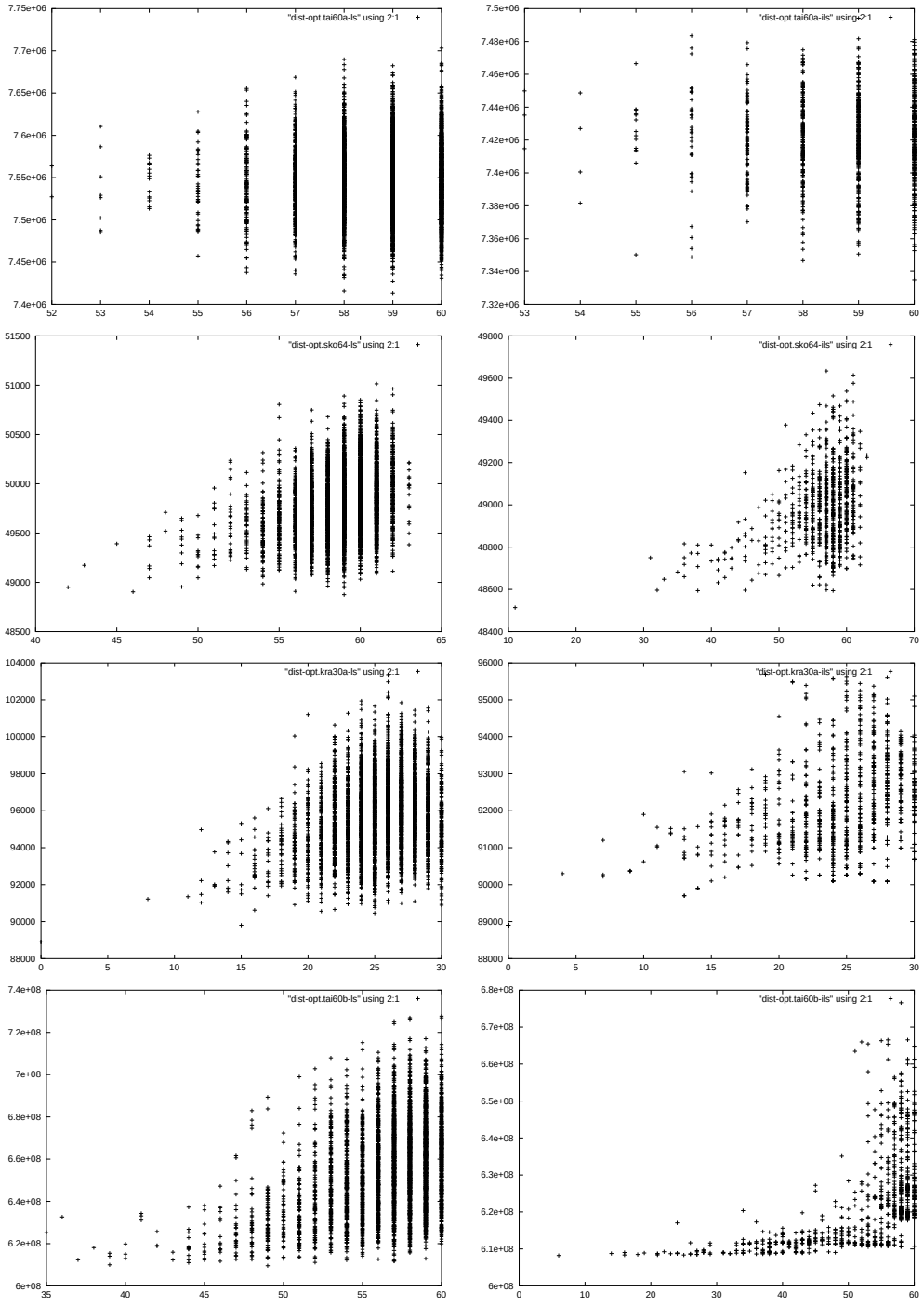

Figure 2: Shown are the Fitness-Distance plots for four QAP instances, from top: `tai60a` (class *i*), `sko64` (class *ii*), kra30a (class *iii*), and `tai60b` (class *iv*). It is given for each instance on the left side a plot corresponding to 5000 2-opt solutions, on the right a plot based on 1000 solutions found by ILS(n). On the x-axis is given the distance to the closest pseudo-optimal solution, on the y-axis the objective function value.

found by using solutions generated by ILS(n) are much higher than the ones observed in experiment E-ls; for these instances also avg^{ils}_{d-opt} is always smaller than avg^{ls}_{d-opt} by a factor of about 0.96, whereas for class i the average distances between solutions in E-ls and E-ils are almost identical. Also the average distance from the pseudo-optimal solutions are somewhat smaller than for the instances of class i. In general, the correlation coefficients for instances of classes iii and iv are higher than those of class ii, which may indicate that these later two classes are somewhat easier to search for ILS algorithms than the instances of class ii. Additionally, one can observe that for the instances with a high flow or distance dominance and high sparsity also a significant FDC can be observed. Hence, these more simpler measures already give a strong indication whether a significant fitness-distance correlation can be expected.

In summary we can conclude that — on average — the better the solution quality the closer a solution is to an optimal solution for the more structured instances of classes $ii - iv$. These instance show a structure in the following sense: The optimal solutions determine the preferred locations of items. Hence, the more locations for items a solution has in common with an optimal solution, the better will be that solution. As we have argued before, such a significant correlation also indicates the potential usefulness of an ILS approach to the QAP.

6 Investigating the Run-time Behavior of ILS for the QAP

In this section we investigate the run-time behavior of the ILS algorithm when applied to the QAP. ILS algorithms are randomized algorithms since they involve random decisions like a randomly generated initial solution and randomly applied mutations. For randomized algorithms the solution quality obtained after running the algorithm for a given time $t_{\max}$ or the time needed to reach a certain required solution cost $c_{\min}$ are random variables. When running a randomized algorithm one is empirically measuring these random variables. Obviously, knowledge on the distribution of these random variables is important to assess the behavior of the algorithm under consideration. Here we adopt the second point of view, where T_c is a random variable with associated distribution function $F(T_c)$ which describes the run-time of the algorithm needed to reach a solution of cost c.

To empirically measure the distribution of T_c for our ILS algorithm, we run it various times up to some maximal run-time $t_{\max}$. Then, for a given required solution cost the empirical cumulative distribution of the algorithm's run-time to find such a solution is determined as

$$\widehat{F}(T_c \leq t) = |\{j \mid rt(j) \leq t \ \wedge \ f(\phi_j) \leq c\}|/l \tag{7}$$

where $rt(j)$ is the run time of the jth run to get a solution quality better than c and l is the total number of runs.

In Figure 3 we present empirical run-time distributions, based on 100 ILS runs, for one instance from each class with respect to different bounds on the solution quality (The parameters were set as $k_{\min} = 3$ and $k_{\max} = 0.9 \cdot n$). The instances tested are `tai30a`

(class i), `sko42` (class ii), `ste36a` (class iii), and `tai35b` (class iv). Additionally, the cumulative distribution function of an exponential distribution (indicated by $ed(x)$ in the plots) which well matches the lower part of the empirical distributions is plotted (its meaning is explained below). Note that in Figure 3 the run-time distributions are plotted to reach a solution within a certain percentage of the best known solution value instead to require a fixed solution cost c.

The empirical run-time distributions show that the ILS algorithm is able to find the pseudo-optimal solutions with a high probability on three of the four instances. An exception is instance `tai30a`, which is very hard for our ILS algorithm. Only in one of 100 runs the best known solution could be found. Yet, it is well known that for this type of instances, local search easily finds solutions within 1 to 2% of the best known one, but optimal solutions are very hard to find [54]. In general, the run-time distributions show that the ILS algorithm consistently finds high quality solutions within 1 or 2% of the best known solution.

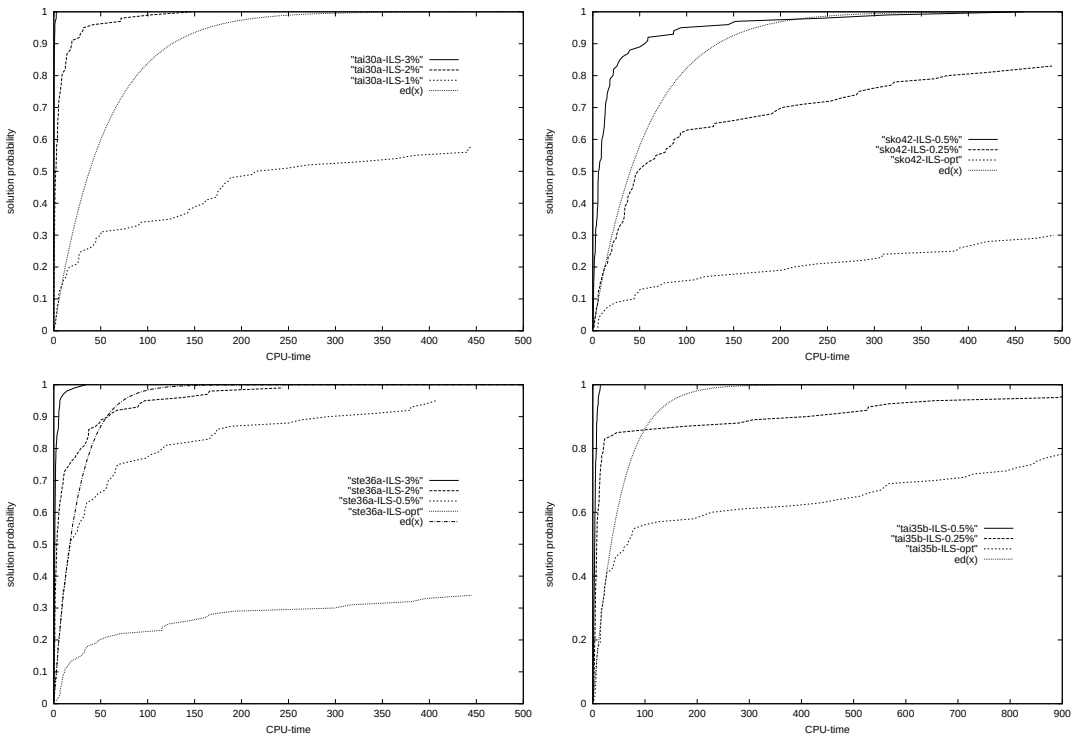

Figure 3: Run-time distributions for QAP instances using iterated `2-opt` with a first-improvement pivoting rule. Given are the distributions for `tai30a` (upper left side), `sko42` (upper right side), `ste36a` (lower left side) and `tai35b` (lower right side). The ILS algorithm uses first-improvement local search and only accepts better quality solutions. The run-time distributions are based on 100 independent runs of each algorithm.

Let us turn now the attention to the importance of the plotted exponential distribution.

Based on a well known theorem from statistical theory [43], if for a given algorithm T_c is distributed exponentially, then the probability of finding a solution when running the algorithm l times for time t is the same as running the algorithm once for time $l \cdot t$. Hence, if the run-time distribution is in fact an exponential, it does not matter whether we restart the algorithm or not; on the long run the solution probability will be the same.

In Figure 3 it can be observed that the empirical run-time distributions for finding very high quality solutions (like the best-known ones) for low run-times well match the exponential distribution, yet for longer run-times they fall strongly below the exponential distributions. To increase the probability of finding a solution for longer run-times, restarting it at an appropriately determined cutoff time would increase the probability of finding such a solution. This is the case since, due to the theorem cited before, the algorithm's probability of finding a solution of the required quality would roughly follow the exponential distribution which matches the initial part of the empirical run-time distribution. In general, the probability of finding a solution of cost c or better when restarting the algorithm k times after computation time t can be calculated as $\hat{p}_{l \cdot t} = 1 - (1 - \hat{p}_t)^l$. To illustrate the possible improvements by restarts, consider the following example. When trying to solve instance `tai35b` to pseudo-optimality, the ILS algorithm reaches after $t = 26$ seconds an empirical solution probability of $\hat{p}_{26} = 0.4$. If restarting the algorithm after that time from a new, randomly generated solution one would obtain after $l = 10$ restarts a solution probability of 0.994. Yet, after $l \cdot t = 260$ seconds the ILS without using restarts only reaches a solution probability of 0.60. Hence, by restarting the algorithm a much higher probability of finding the best known solution can be obtained which is also shown for this particular example in Figure 4. In summary, the ILS algorithms show a type of stagnation behavior since, ideally, the development of the probability for finding the required solution quality should follow the exponential distribution.

One interpretation of the stagnation behavior is that the ILS algorithm is not capable of efficiently exploring regions of the search space which are at a larger distance from the current solution. A somewhat limited exploration of more distant solutions from the current search point is given by increased values for k, yet this mechanism alone appears to be too weak, since for large values of k the mutated solution would be rather similar to a randomly generated solution. Instead, a better idea appears to be to explore more efficiently distant regions of the search space by allowing the algorithm to move away from the current search point or to initially explore different regions of the search space by a population of search points and then to concentrate the search on the most promising ones identified so far. In the next section we present such possibilities of extended acceptance criteria and population-based extensions.

7 ILS Extensions

In this section we propose extensions of the basic ILS algorithm which has been presented in Section 3. The proposed extensions include the use of acceptance criteria which allow moves to worse solutions as well as population-based extensions of ILS algorithms. The

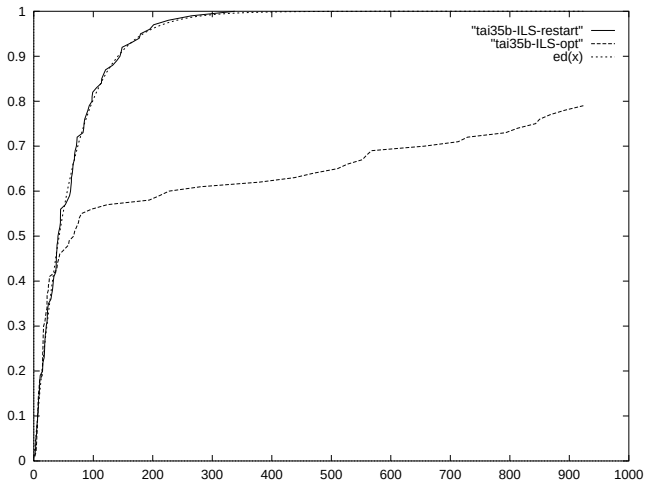

Figure 4: Run-time distributions for QAP instance using iterated `2-opt` with a first-improvement pivoting rule (ILS) and a restart version of ILS (ILS-restart) on instance `tai35b`. The restart algorithm uses a fixed cutoff of 26 seconds. The run-time distributions are based on 100 independent runs of each algorithm.

computational results which we present in the next section will show that these extensions may find significantly improved computational results compared to the basic ILS algorithm.

7.1 ILS with extended Acceptance Criteria

As shown by the analysis of the run-time distributions, the ILS algorithm for the QAP can be substantially improved by providing a stronger exploration of the search space. In general, a stronger exploration of the search space can be easily incorporated into the acceptance criterion by accepting moves to worse solutions.

In the simplest case one can restart the algorithm from a new randomly chosen solution, we call such an acceptance criterion $Restart(s, s'', history)$. In $Restart(s, s'', history)$ we use a *soft* restart which is related the algorithm's search progress: If the ILS algorithm does not find an improved solution for a fixed number of iterations it_0, we assume that it is stuck and restart it from a new, randomly generated solution. Note that counting the number of iterations since the last improved solution has been found can be interpreted as a very simple use of the search history.

Exploration of the search space is very much favored if s'' is always accepted; this acceptance criterion is called $RandomWalk(s, s'')$ in the following. A bias towards better solutions can be introduced by accepting worse solutions only with a certain probability. This is achieved, for example, by a simulated annealing type criterion which also has been used in [34, 28] for applications of ILS to the TSP and the job shop scheduling problem, respectively. We denote this acceptance criterion $LSMC(s, s'')$, reminiscent of the term

large step Markov chains used for one of the first ILS algorithms [34]. In particular, s'' is accepted with probability p_{accept} given by:

$$p_{accept}(Temp, s, s'') = \begin{cases} 1 & \text{if } f(s'') < f(s) \\ \exp(\frac{f(s)-f(s'')}{Temp}) & \text{otherwise} \end{cases} \quad (8)$$

Temp is a parameter called temperature and it is lowered during the run of the algorithm according to a temperature schedule like in Simulated Annealing. In fact, when applying $LSMC(s, s'')$, the temperature schedule we use is a non-monotonic temperature schedule as proposed in [20] for simulated annealing. If the acceptance rate of worse solutions is very low, we increase *Temp* to its initial value.

7.2 Population-based ILS Extensions

Our motivation for using a population of solutions is to investigate other means of avoiding the stagnation behavior of ILS than by increasing search space exploration in single search point ILS. The idea is to replace one single search point by a population of search points which may interact in some way. In population-based ILS extensions the stagnation behavior is avoided by offering at the start a number of different solutions and, in some sense, delaying the decision on which solution one has to concentrate to find the highest solution quality. Each of the search points follows the basic ILS algorithm outlined in Section 3. Here we propose two simple schemes how ILS algorithms can be extended into this direction, but certainly many others possible population-based extensions can be conceived.

Population-based ILS extensions share similarities to other population-based search metaphors like evolutionary algorithms [1], in particular to those of genetic local search [39, 40, 36]. In genetic local search solutions are modified by mutation operators applied to single solutions and by crossover operators which exchange information between solutions. To the so generated solutions local search is applied. A deliberate difference between the proposed population-based ILS extensions to genetic local search is that in ILS solutions are only modified by applications of a kick-move, which corresponds to mutations in the context of evolutionary algorithms.

Replace-Worst In this variant we use a population of individual ILS runs which has only very limited interactions. We start with a number of μ ILS runs. Each of the ILS applications is run independently of the others, except that occasionally a copy of the current best solution replaces the worst solution of the population. This replacement takes place every r_b iterations. The motivation for applying this scheme is to gradually concentrate the search around the best solution of the population. The parameter r_b determines how fast this concentration takes place. If r_b is chosen very low, the search may concentrate very early around possibly suboptimal solutions, if r_b is chosen very high, replace-worst will behave very similar to multiple independent runs of an ILS algorithm.

Evolution Strategies Additionally we propose a population-based extension which replaces the current population using the $(\mu + \lambda)$-selection stemming from evolution strategies [1]. The population is of size μ and we generate in each algorithm iteration λ new local optima by λ times randomly choosing a solution and applying to it one iteration of the basic ILS algorithm. The new set of solutions in the $(\mu + \lambda)$-selection is then simply taken as the μ best out of the $\mu + \lambda$ possible solutions. Since this way of population manipulation very strongly favours the best solutions, care has to be taken that the population does not converge too early. Therefore, we apply the following scheme of generating the new population of solutions. First, the $\mu + \lambda$ solution are sorted according to the their solution cost and solutions are considered for insertion in the population in that order. A solution is inserted if the distance to any of the already inserted solutions into the population is larger than some minimal distance $d_{\min}$. In fact, we vary $d_{\min}$ dynamically during the run of the algorithm by subsequently lowering the actual value of $d_{\min}$ during the run. The aim is to keep the solutions at the start of the algorithm at a rather high distance to explore sufficiently distant regions of the search space.

8 Experimental Results

In this section we experimentally evaluate the performance of the proposed ILS algorithms on a wide range of QAP instances from QAPLIB. To put the computational results obtained with the ILS algorithms into perspective, we also give computational results for Robust Tabu Search [52] (`Ro-TS`) and an Ant Colony Optimization algorithm presented in [50] ($\mathcal{MM}$AS). These two algorithms have been chosen since `Ro-TS` is among the best algorithms for problem classes *i* and *ii* [54, 15] while the $\mathcal{MM}$AS algorithm has been shown to be among the best available algorithms on structured instances and, in particular, those of class *iv* [50]. All algorithms are given the same computation time (indicated by t_{max} in Tables 3 and 4) which corresponds to the time needed by `Ro-TS` to do $1000 \cdot n$ iterations. The computation times are measured as seconds on a SUN UltraSparc II machine with two UltraSparc I processors (167Mhz) with 0.5MB external cache and 256 MB main memory. Only one single processor has been used due to the sequential implementation of the algorithms. All algorithms use the same local search implementation, hence, the comparison is fair in this respect. As indicated in Section 3, for symmetric instances the calculation of the effect of exchanging the location of two items can be calculated faster than for asymmetric instances which explains the differences for the time limit t_{max} for instances of the same size.

We present computational results with the standard ILS algorithm as described in Section 3 (referred to as `Better` in the following), and with the proposed extensions. We refer to the extensions as follows: to the ILS algorithm using acceptance criterion $LSMC(s, s'')$ as `LSMC`, to the one using acceptance criterion $RandomWalk(s, s'')$ as `RW`, and to the one using acceptance criterion $Restart(s, s'', history)$ as `Restart`. To the population-based extensions we refer to as `RepWorst` for the first and as `ES` to the second extension.

The parameters for the ILS variants were set as follows. In `Better` we set $k_{\min} = 3$ and $k_{\max} = 0.9 \cdot n$; the same parameter settings are used in variant `RW`. These are the only parameters for these two algorithms. In `Restart` we use the same settings for $k_{\min}$ and $k_{\max}$

as in `Better`and set $u_b = 2.5 \cdot k_{\max}$. For variant `LSMC` an annealing schedule has to be defined. We have chosen it in a straightforward way as follows: After the initial solution ϕ_0 is locally optimized to yield solution ϕ', we set $Temp_{init} = 0.025 \cdot f(\phi')$ (a solutions which is 2.5% worse than the current one is accepted with probability $1/e$). The temperature is lowered every 10 iterations according to a geometric cooling scheme, by setting $Temp_{i+1} = 0.9 \cdot Temp_i$. If during the last 100 iterations of the algorithm less than three times a worse solution has been accepted, the temperature is reset to T_{init}. In `LSMC` we set $k_{\max} = \max\{0.9 \cdot n, 50\}$, since the search space should be explored by occasionally accepting worse solutions and not by very high values for k. In the population-based extensions the parameters are set as follows: We use a population size of 30 for both, `RepWorst` and `ES`, and in both we set $k_{\max} = 10$. In `RepWorst` in the first 30 iterations the ILS runs are completely independent from each other and from then on we set $r_b = 3$. In `ES` in every iteration, like in `RepWorst`, 30 new solutions are generated, i.e., $\mu = \lambda = 30$. At the start of `ES` we set $d_{\min} = 2/3 \cdot n$ and lower it in the following iterations to $d_{\min} = \max\{5, 2/3 \cdot n - it\}$, where it is the iteration counter. In both algorithms we apply a search diversification if the average distance in the current population is smaller than 10; it consists in applying four iterations of `RW` to each solution using a mutation of size $k = n/2$. Additionally, in the variants `LSMC`, `RepWorst`, and `ES` we initially set $k_{\min} = k_{\max}$ and from then on $k_{\min} = \min\{3, k_{\max} - it\}$. When in `LSMC` the temperature is reset to T_{init} or diversification takes place in the population-based extensions we also reset $k_{\min}$ as described before (by resetting also it to zero).

The computational results are given in Table 3 for the instances of class *i* and *ii* and in Table 4 for the instances of class *iii* and *iv*. A first conclusion which can be drawn from the computational results is that, in general, with the proposed extensions considerable improvements over `Better` are possible. For example, the very straightforward extension given by `Restart` achieves better average solution qualities on all instances. Overall, `LSMC` and the two population-based extensions are the best performing ones. Yet, whether or how much does an ILS extension improves the performance of `Better` strongly depends on the problem class.

For the unstructured QAP instances of class *i* and the instances of class *ii* (see Table 3) all extensions show a significantly improved performance over `Better`. Here, the best performing extensions are `ES` and `LSMC`, closely followed by `RepWorst` and `RW`. On the more strongly structured instances of class *iii* and *iv* the relative performance of the ILS algorithms is different. Here, `ES`, `RepWorst`, `Restart`, and `LSMC` are the best ones. On most of these instances even `Better` performs better than `RW`, while `RW` gives significantly better results than ILS on instances of classes *i* and *ii*. This fact can be explained to some extent with the different structure of these instances as shown in Section 5. While instances of class *i* do not show any or only a very weak fitness-distance correlation, for instances of class *iii* and *iv* the fitness-distance correlation is very high. Hence, intuitively, on the later type of instances a bias towards the best solutions found during the search, as done in `Better`, appears to be plausible. Since `RW` does not exploit this structure of these QAP instances, it does not reach the performance of `Better` on most of them.

Somewhat surprising is the good performance of `RW` on some of the structured instances and the instances of class *ii* that also show a significant fitness-distance correlation (although

Table 3: Experimental results for ILS and several extensions on QAP instances taken from QAPLIB for unstructured instances. Given is the percentage deviation from the best known solution over 10 independent runs of the algorithms. Additionally we have computed the average percentage deviation for each algorithm on the specific problem class, indicated by Average. See the text for a description of the variants used.

Problem instance	Better	LSMC	RW	Restart	RepWorst	ES	Ro-TS	$\mathcal{MM}$AS	t_{max}
random problems with entries uniformly distributed									
tai20a	0.723	0.503	0.542	0.467	0.500	0.344	**0.108**	0.428	9
tai25a	1.181	0.876	0.896	0.823	0.869	0.656	**0.274**	1.751	17
tai30a	1.304	0.808	0.989	1.141	0.707	0.668	**0.426**	1.286	30
tai35a	1.731	1.110	1.113	1.371	1.010	0.901	**0.589**	1.568	51
tai40a	2.036	1.319	1.490	1.491	1.305	1.082	**0.990**	1.131	75
tai50a	2.127	1.496	1.491	1.968	1.574	1.211	**1.125**	1.900	150
tai60a	2.200	1.498	1.692	2.081	1.622	1.349	**1.203**	2.484	265
tai80a	1.775	1.198	1.200	1.576	1.219	1.029	**0.900**	2.103	670
random flows on grids									
nug30	0.219	0.020	0.052	0.020	0.013	**0.007**	0.013	0.042	30
sko42	0.269	0.010	0.010	0.161	0.002	**0.0**	0.025	0.104	92
sko49	0.226	0.133	0.133	0.139	0.090	**0.068**	0.076	0.150	135
sko56	0.418	0.087	0.087	0.153	0.102	**0.071**	0.088	0.118	211
sko64	0.413	0.068	0.068	0.202	0.079	**0.057**	0.071	0.243	308
sko72	0.383	0.134	0.134	0.294	0.139	**0.085**	0.146	0.243	455
sko81	0.586	0.101	0.100	0.194	0.100	**0.082**	0.136	0.223	656
sko90	0.576	0.131	0.187	0.322	0.262	**0.128**	**0.128**	0.288	895
sko100a	0.358	0.115	0.161	0.257	0.191	**0.109**	0.128	0.191	1240

slightly smaller than those of class *iii* and *iv*). Part of this success of RW may be explained by the large number of local searches which can be done in the same computation time compared to Better. For example, in the given computation time with RW on instance tai80b roughly 5000 times local search can be applied, while Better in the same run-time can only apply on average 1600 times a local search. This difference is mainly due to the fact, that for RW the value of k, which determines the strength of the mutations, is always very small. To illustrate this point in Figure 5 the development of k versus the iteration counter is plotted for Better and RW on instance tai80b. This point is important since due to the reseting strategy of the don't look bits (described in Section 3) at small values of k the local search runs much faster; the time needed for each iteration of RW is on average much smaller than for Better. In Figure 5 it can also be observed that Better may find an improved solution at rather high values for k (see the peak at *it* around 350 which corresponds to $k = 51$). In fact, if $k_{\max}$ is chosen too low in Better, its performance on many instances decreases considerably.

When compared to Ro-TS and $\mathcal{MM}$AS the ILS extensions show a particularly good performance on the structured instances. Among the algorithms compared, Ro-TS is the best on the instances of class *i* and shows very good performance on the instances of class *ii*, slightly worse than ES and comparable to LSMC. Yet, on class *iii* and *iv* the best four ILS

Table 4: Experimental results for ILS and several extensions on QAP instances taken from QAPLIB for structured instances. See Table 5.7 for a description of the entries.

Problem instance	Better	LSMC	RW	Restart	RepWorst	ES	Ro-TS	$\mathcal{MM}$AS	t_{max}
real life instances									
bur26a-h	**0.0**	0.001	0.001	**0.0**	**0.0**	**0.0**	0.002	**0.0**	44
chr25a	3.862	2.745	9.521	1.860	**1.264**	1.475	7.550	1.827	18
kra30a	0.672	0.090	**0.0**	0.134	**0.0**	**0.0**	0.268	0.314	30
kra30b	0.094	0.026	0.046	0.051	0.031	**0.008**	0.023	0.049	30
ste36a	0.377	0.099	0.451	0.227	0.071	**0.015**	0.155	0.181	54
ste36b	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	0.081	**0.0**	54
randomly generated real-life like instances									
tai20b	0.045	**0.0**	0.045	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	10
tai25b	**0.0**	**0.0**	0.007	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	41
tai30b	**0.0**	**0.0**	0.093	**0.0**	**0.0**	**0.0**	0.107	**0.0**	73
tai35b	0.131	0.049	0.081	**0.0**	**0.0**	**0.0**	0.064	**0.0**	117
tai40b	**0.0**	**0.0**	0.204	**0.0**	**0.0**	**0.0**	0.531	**0.0**	177
tai50b	0.203	0.185	0.282	0.028	0.042	0.033	0.342	**0.002**	348
tai60b	0.029	0.059	0.645	0.023	0.005	**0.0**	0.417	0.005	633
tai80b	0.785	0.256	0.703	0.260	0.222	0.383	1.031	**0.096**	1510
tai100b	0.219	0.096	0.711	0.202	0.113	**0.083**	0.512	0.142	2910

algorithms obtain much better results than Ro-TS. For the instances of class *iv* $\mathcal{MM}$AS is overall the best algorithm of those compared, but it is very closely followed by the two population-based extensions, Restart and LSMC. In summary, the comparison of the ILS algorithms to state-of-the art algorithms shows that, despite their conceptual simplicity, they belong to the best algorithms for structured QAP instances.

9 Discussion and Conclusions

In this article we have presented and analyzed an iterated local search (ILS) algorithm applied to the quadratic assignment problem. The proposed ILS algorithm applies random mutations of varying strength to the best solution found so far and subsequently applies a fast local search procedure. The local search uses a technique called don't look bits which has been adapted from applications of local search algorithms to the TSP. Independent from our ILS approach in [55] an ILS algorithm similar to the one presented in Section 3 has been implemented. That ILS algorithm gives roughly comparable results to our approach. Yet, our work extensively analyzes and strongly improves the initially proposed ILS algorithm which has not been done in [55].

We have justified the application of an ILS algorithm to the QAP by a search space analysis of the QAP. This analysis has shown that, depending on the particular QAP instance type, a strong correlation between the solution quality and the distance to the closest global optimum exists. In particular, a large fitness-distance correlation is observed on structured, real-life QAP instances while for randomly generated instances according to uniform distri-

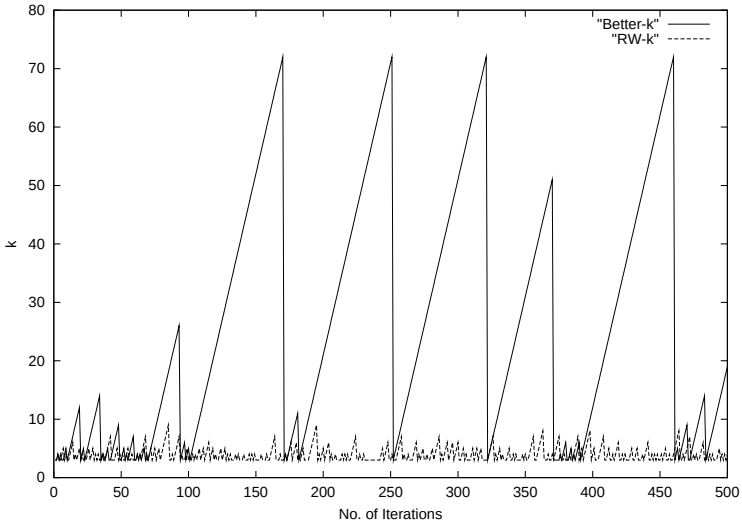

Figure 5: Development of the kick-move strength for `Better` and `RW`.

butions such a correlation does not exist or it is very weak.

We analyzed the run-time behavior of the proposed ILS algorithm if high solution quality is required. This analysis has shown that the algorithm is able to reach high quality solutions in very short time. Yet, when longer run-times are allowed, the search progress stagnates. Based on this analysis we proposed ILS extensions which use acceptance criteria allowing to continue the search from worse solutions as well as population-based extensions in which the single search point of an ILS algorithm is replaced by a population of search points.

In an experimental analysis of the proposed ILS algorithms we have shown that the extensions improve strongly the obtained solution quality of the most basic ILS algorithm. Additionally the proposed ILS extensions have the advantage that they are very easy to implement. Our comparison to other state-of-the-art algorithms for the QAP has shown that the proposed ILS extensions are currently among the top performing algorithms for structured QAP instances.

Run-time distributions have been observed occasionally in the literature [52, 2, 53, 57], but they have been mainly used for simple descriptive purposes or to get hints on the obtainable speed-up for parallel processing based in multiple independent runs of a sequential algorithm. In [53] general conditions are given when parallel independent runs on p processors of an algorithm for time t give better performance than a sequential algorithm run for computation time $p \cdot t$. Yet, there it is argued that these conditions are hardly met in praxis. A novel use of run-time distributions in our work is that we use empirically measured run-time distributions to get hints on situations in which ILS algorithms can be improved. Based on their observation we propose ILS extensions which significantly improve the performance. It is important to note that the proposed ILS extensions are problem independent

and therefore they can be used to improve ILS algorithms for other problems. A common feature of all proposed ILS extensions which use no population is that they allow moves to worse solutions by the acceptance criterion. Therefore, our results also suggest that the appropriate choice of the acceptance criterion should receive more attention in future ILS applications.

There are several possible ways to extend the presented work. One possibility is to further improve the computational results by a more fine-tuned implementation. One such possibility is to make stronger use of the history component which is indicated in the general algorithmic outline of ILS given in Figure 1. Such extensions could strongly benefit from the use of long-term memory techniques used in tabu search [17]. An interesting research direction arises from the observation of the stagnation behavior observed in the run-time behavior of the ILS algorithm. In [2, 52] it has been observed that for two specific tabu search algorithms for the QAP, one of these is the robust tabu search algorithm applied in Section 8, the required run-time to find the pseudo-optimal solution for unstructured instances of type i (defined in Section 4) follows an exponential distribution. Yet, we have evidence that ILS algorithms applied also to other problems like the TSP do suffer from the stagnation behavior shown here [49]. Hence, further research has to be done to investigate which kinds of algorithms do show stagnation behavior and which features of an algorithm or of the problem to be solved lead to such a behavior.

Acknowledgments

I would like to thank Holger Hoos for discussions on the use of run-time distributions. Large part of this research has been done at the TU Darmstadt, FG Intellektik. Part of this work has been supported by a Madame Curie Fellowship awarded to Thomas Stützle (CEC-TMR Contract No. ERB4001GT973400).

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[2] R. Battiti and G. Tecchiolli. Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessor and Microsystems*, 16(7):351–367, 1992.

[3] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[4] J.L. Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

[5] K.D. Boese. *Models for Iterative Global Optimization.* PhD thesis, University of California, Computer Science Department, Los Angeles, 1996.

[6] K.D. Boese, A.B. Kahng, and S. Muddu. A New Adaptive Multi-Start Technique for Combinatorial Global Optimization. *Operations Research Letters*, 16:101–113, 1994.

[7] R.E. Burkard and J. Offermann. Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme. *Zeitschrift für Operations Research*, 21:B121–B132, 1977.

[8] R.E. Burkard and F. Rendl. A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems. *European Journal of Operational Research*, 17:169–174, 1984.

[9] D.T. Connolly. An Improved Annealing Scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[10] V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares. A Scatter Search Based Approach for the Quadratic Assignment Problem. In T. Baeck, Z. Michalewicz, and X. Yao, editors, *Proceedings of ICEC'97*, pages 165–170. IEEE Press, 1997.

[11] J.W. Dickey and J.W. Hopkins. Campus Building Arrangement Using TOPAZ. *Transportation Science*, 6:59–68, 1972.

[12] H.A. Eiselt and G. Laporte. A Combinatorial Optimization Problem Arising in Dartboard Design. *Journal of the Operational Research Society*, 42:113–118, 1991.

[13] A.N. Elshafei. Hospital Layout as a Quadratic Assignment Problem. *Operations Research Quarterly*, 28:167–179, 1977.

[14] C. Fleurent and J.A. Ferland. Genetic Hybrids for the Quadratic Assignment Problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 173–187. American Mathematical Society, 1994.

[15] L.M. Gambardella, É.D. Taillard, and M. Dorigo. Ant Colonies for the QAP. Technical Report IDSIA-4-97, IDSIA, Lugano, Switzerland, 1997. Accepted for publication in the *Journal of the Operational Research Society (JORS)*.

[16] A.M. Geoffrion and G.W. Graves. Scheduling Parallel Production Lines with Changeover Costs: Practical Applications of a Quadratic Assignment/LP Approach. *Operations Research*, 24:595–610, 1976.

[17] F. Glover. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. In Barr, Helgason, and Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.

[18] P. Hansen and N. Mladenović. Variable Neighbourhood Search for the p-Median. Technical Report Les Cahiers du GERAD G-97-39, GERAD and École des Hautes Études Commerciales, 1997.

[19] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Boston, 1999.

[20] T.C. Hu, A.B. Kahng, and C.-W.A. Tsao. Old Bachelor Acceptance: A New Class of Non-Monotone Threshold Accepting Methods. *ORSA Journal on Computing*, 7(4):417–425, 1995.

[21] D.S. Johnson and L.A. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[22] T. Jones and S. Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In L.J. Eshelman, editor, *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufman, 1995.

[23] S.A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.

[24] J. Krarup and P.M. Pruzan. Computer-aided Layout Design. *Mathematical Programming Study*, 9:75–94, 1978.

[25] G. Laporte and H. Mercure. Balancing Hydraulic Turbine Runners: A Quadratic Assignment Problem. *European Journal of Operational Research*, 35:378–381, 1988.

[26] Y. Li, P.M. Pardalos, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.

[27] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[28] H. Ramalhinho Lourenço. Job-Shop Scheduling: Computational Study of Local Search and Large-Step Optimization Methods. *European Journal of Operational Research*, 83:347–364, 1995.

[29] H. Ramalhinho Lourenço and M. Zwijnenburg. Combining the Large-Step Optimization with Tabu-Search: Application to the Job-Shop Scheduling Problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 219–236. Kluwer Academic Publishers, 1996.

[30] V. Maniezzo. Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. Technical Report CSR 98-1, Science dell'Informazione, Universitá di Bologna, Sede di Cesena, 1998.

[31] V. Maniezzo and A. Colorni. The Ant System Applied to the Quadratic Assignment Problem. Accepted for publication in *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[32] V. Maniezzo, M. Dorigo, and A. Colorni. The Ant System Applied to the Quadratic Assignment Problem. Technical Report IRIDIA/94-28, Université de Bruxelles, Belgium, 1994.

[33] O. Martin and S.W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[34] O. Martin, S.W. Otto, and E.W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991.

[35] O. Martin, S.W. Otto, and E.W. Felten. Large-step Markov Chains for the TSP Incorporating Local Search Heuristics. *Operations Research Letters*, 11:219–224, 1992.

[36] P. Merz and B. Freisleben. A Genetic Local Search Approach to the Quadratic Assignment Problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA'97)*, pages 465–472. Morgan Kaufmann, 1997.

[37] P. Merz and B. Freisleben. Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. Technical Report TR-98-01, University of Siegen, Department of Electrical Engineering and Computer Science, 1998.

[38] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24:1097–1100, 1997.

[39] H. Mühlenbein. Evolution in Time and Space – the Parallel Genetic Algorithm. In *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann Publishers, 1991.

[40] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution Algorithms in Combinatorial Optimization. *Parallel Computing*, 7:65–85, 1988.

[41] V. Nissen. Solving the Quadratic Assignment Problem with Clues from Nature. *IEEE Transactions on Neural Networks*, 5(1):66–72, 1994.

[42] C. Reeves. Landscapes, Operators and Heuristic Search. Accepted for publication in *Annals of Operations Research*, 1999.

[43] V.K. Rohatgi. *An Introduction to Probability Theory and Mathematical Statistics*. John Wiley & Sons, 1976.

[44] S. Sahni and T. Gonzalez. P-Complete Approximation Problems. *Journal of the ACM*, 23(3):555–565, 1976.

[45] J. Skorin-Kapov. Tabu Search Applied to the Quadratic Assignment Problem. *ORSA Journal on Computing*, 2:33–45, 1990.

[46] P.F. Stadler. Towards a Theory of Landscapes. Technical Report SFI–95–03–030, Santa Fe Institute, 1995.

[47] L. Steinberg. The Backboard Wiring Problem: A Placement Algorithm. *SIAM Review*, 3:37–50, 1961.

[48] T. Stützle. Applying Iterated Local Search to the Permutation Flow Shop Problem. Technical Report AIDA–98–04, FG Intellektik, TU Darmstadt, August 1998.

[49] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, FB Informatik, TU Darmstadt, 1998.

[50] T. Stützle and M. Dorigo. ACO Algorithms for the Quadratic Assignment Problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*. McGraw Hill, 1999. Also available as Technical Report IRIDIA/99-02, Université de Bruxelles, Belgium, 1999.

[51] T. Stützle and H.H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and Local Search for Combinatorial Optimization Problems. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 137–154. Kluwer, Boston, 1999.

[52] É.D. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

[53] É.D. Taillard. Parallel Tabu Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[54] É.D. Taillard. Comparison of Iterative Searches for the Quadratic Assignment Problem. *Location Science*, 3:87–105, 1995.

[55] É.D. Taillard and L.M. Gambardella. Adaptive Memories for the Quadratic Assignment Problem. Technical Report IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.

[56] D.M. Tate and A.E. Smith. A Genetic Approach to the Quadratic Assignment Problem. *Computers & Operations Research*, 22(1):73–83, 1995.

[57] H.M.M. ten Eikelder, M.G.A. Verhoeven T.V.M. Vossen, and E.H.L. Aarts. A Probabilistic Analysis of Local Search. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: Theory & Applications*, pages 605–618. Kluwer, Boston, 1996.

[58] E.D. Weinberger. Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. *Biological Cybernetics*, 63:325–336, 1990.