Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 174 (2006) 1519–1539
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Iterated local search for the quadratic assignment problem

Thomas Stützle

FG Intellektik, FB Informatik, TU Darmstadt, Hochschulstr. 10, 64283 Darmstadt, Germany

Received 7 May 2001; accepted 10 January 2005
Available online 13 May 2005

Abstract

Iterated local search (ILS) is a simple and powerful stochastic local search method. This article presents and analyzes the application of ILS to the quadratic assignment problem (QAP). We justify the potential usefulness of an ILS approach to this problem by an analysis of the QAP search space. However, an analysis of the run-time behavior of a basic ILS algorithm reveals a stagnation behavior which strongly compromises its performance. To avoid this stagnation behavior, we enhance the ILS algorithm using acceptance criteria that allow moves to worse local optima and we propose population-based ILS extensions. An experimental evaluation of the enhanced ILS algorithms shows their excellent performance when compared to other state-of-the-art algorithms for the QAP.

Keywords: 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 [41], campus and hospital layout [13,16], typewriter keyboard design [9] and many others can be formulated as QAPs. The QAP can best be described as the problem of assigning a set of items to a set of locations with given distances between the locations and given flows between the items. The goal is to place the items on locations in such a way that

E-mail address: stuetzle@informatik.tu-darmstadt.de

the sum of the product between flows and distances is minimal.

Given n items and n locations, two $n \times n$ matrices A and B, where a_{ij} is the distance between locations i and j and b_{rs} is the flow between items r and s, the objective in the QAP is to find an assignment of items to locations such that every item is assigned to exactly one location and no location is assigned more than one item. Since in the QAP the number of items is the same as the number of locations, such an assignment corresponds to a permutation of the integers in $\{1, \ldots, n\}$. The objective for the QAP can then be formulated as

doi:10.1016/j.ejor.2005.01.066

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

where Φ is the set of all permutations of $\{1, \ldots, n\}$, and ϕ_i gives the location of item i in a solution $\phi \in \Phi$.

The QAP is an $\mathcal{NP}$-hard optimization problem [38]. It is considered as one of the hardest optimization problems as the largest instances that can be solved today with exact algorithms have instance size around 30 [1,20]; in fact, the largest non-trivial instance from QAPLIB, a benchmark library for the QAP, ever being solved to optimality has 36 locations [8]. Only rarely larger but specially structured instances have been solved by exact algorithms [15]. In practice, the only feasible way to solve large QAP instances is to apply heuristic algorithms which find very high quality solutions in short computation time. Several such algorithms have been proposed which include algorithms like simulated annealing [11], tabu search [4,39,47], memetic algorithms [14,17,33], ant algorithms [19,30,43], and scatter search [12].

In this article, we investigate the performance of iterated local search (ILS) [29] on the QAP. ILS is a very simple and powerful stochastic local search method that has proved to be among the best performing approximation algorithms for the well-known Traveling Salesman Problem (TSP) [25] and a number of other problems [29]. The essential idea of ILS is to perform a biased, randomized walk in the space of locally optimal solutions instead of sampling the space of all possible candidate solutions. This walk is build by iteratively applying first a perturbation to a locally optimal solution, then applying a local search algorithm, and finally using an acceptance criterion which determines to which locally optimal solution the next perturbation is applied.

This article contains several further contributions. First, we present results of an analysis of the QAP search space which indicates that, especially for high quality solutions, in structured and real life QAP instances a significant correlation exists between the solution cost and the distance to optimal solutions. This result intuitively justifies a basic ILS approach, in which the acceptance criterion only accepts better solutions. However, a following analysis of the run-time behavior of such an ILS algorithm shows that it suffers from a stagnation behavior that severely compromises its performance for long runs. Therefore, as suggested by the run-time analysis, we introduce variants of the initial ILS algorithm including new, population-based ones that significantly improve ILS performance. Computational results show that the best performing variants obtain very high performance, comparable to or better than several state-of-the-art algorithms for the QAP.

The paper is structured as follows. In Section 2, we present the details of the initial ILS application to the QAP. Section 3 gives the results of a search space analysis of the QAP. In Section 4 we present an analysis of the run-time behavior of the initial ILS approach and based on the results of this analysis we propose several extensions to the initial ILS algorithm. These extensions are then compared in Section 5 to state-of-the-art algorithms for the QAP and Section 6 contains some concluding remarks.

2. Iterated local search

Iterated local search (ILS) is a simple and generally applicable stochastic local search method that iteratively applies local search to perturbations of the current search point, leading to a randomized walk in the space of local optima [29]. To apply an ILS algorithm, four procedures have to be specified: GenerateInitialSolution generates the starting point of this walk; Perturbation generates new starting points of the local search by perturbing some solution; the AcceptanceCriterion decides from which solution the walk is continued; the LocalSearch procedure implements the local search and also defines (by the solutions it generates as its output) the space in which the walk actually takes place. Fig. 1 gives an algorithmic scheme for ILS. The *history* component in Perturbation and AcceptanceCriterion indicates that also the search history may influence the decisions made in these procedures. Yet, often Markovian implementations of ILS are applied, that is, the output of Perturbation and AcceptanceCriterion is independent of the search history.

```
**procedure** *Iterated Local Search*
    $s_0 \leftarrow$ GenerateInitialSolution
    $s \leftarrow$ LocalSearch($s_0$)
    **repeat**
        $s' \leftarrow$ Perturbation($s$, *history*)
        $s'' \leftarrow$ LocalSearch($s'$)
        $s \leftarrow$ AcceptanceCriterion($s, s''$, *history*)
    **until** termination condition met
**end** *Iterated Local Search*
```

Fig. 1. General algorithmic outline for an iterated local search method.

ILS is conceptually a rather simple stochastic local search method. This is due to the simple 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. Despite its simplicity, it is at the basis of several state-of-the-art algorithms for problems like the TSP [25] or scheduling problems [2,10]. Probably the conceptual simplicity of ILS also led to the phenomenon that its general idea has been rediscovered by many authors and has lead to many different names for ILS like *iterated descent* [5], *large-step Markov chains* [32], *chained local optimization* [31] etc. Nevertheless, the term iterated local search now becomes widely accepted [29].

To apply ILS to the QAP, the four component procedures have to defined. In our "standard" ILS approach these procedures are defined as follows.

GenerateInitialSolution

As the initial solution we use a random assignment of items to locations, mainly because high performing construction heuristics for the QAP are not known.

LocalSearch

We apply an iterated descent algorithm for LocalSearch, which uses the 2-opt neighborhood like many other local search approaches to the QAP: The neighborhood $\mathcal{N}(\phi)$ of a solution ϕ is defined by the set of permutations which can be obtained by exchanging two items r and s at positions ϕ_r and ϕ_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 the location of two items r and s can be computed in O(n) [48].

Our iterated descent algorithm uses a first-improvement pivoting rule: once an improving move is found, it is immediately applied. A disadvantage of the first-improvement algorithm is that every full neighborhood scan has a complexity of $\mathcal{O}(n^3)$. To avoid this, we adopted the technique of *don't look bits*, initially proposed to speed up local search algorithms for the TSP [6,32], to the QAP: When applied to the QAP, a don't look bit is associated with every item. When starting the local search, all don't look bits are turned off (set to 0). If during the neighborhood scan for an item no improving move is found, the don't look bit is turned on (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.[1]

Perturbation

The Perturbation exchanges k randomly chosen items, corresponding to a random move in the k-opt neighborhood. In initial experiments (not reported here), we tested several fixed values for k. We found that the best parameter setting for k was rather dependent on the particular instance under solution. To make the particular choice of the perturbation 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 finally decided to adapt k as done in variable neighborhood search [21]. We vary k between two values k_{min} and k_{max} starting at k_{min}: If after the perturbation and the subsequent local search no better solution is found, k is increased by one; otherwise it is set to k_{min}. If we have $k = k_{max}$, we set k back to k_{min}.

In the procedure Perturbation one can easily exploit the don't look bits: only the don't look bits of

[1] Note that this technique was proposed already for the first version of this paper that was published as a Technical Report [46] and it was adopted independently also in [33] to the QAP.

items which change their location due to the perturbation are reset to 0. This resetting strategy of the don't look bits results in a very significant speed improvement of the local search algorithm at only a minor loss of solution quality, as we could verify in preliminary experiments.

AcceptanceCriterion

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

$$s \leftarrow Better(s, s'') = \begin{cases} s'' & \text{if } f(s'') < f(s), \\ s & \text{otherwise}, \end{cases} \tag{2}$$

where $f(s)$ is the objective function value of solution s. This choice of the acceptance criterion appears to be the standard when applying ILS algorithms [21,25,26,31] and has the advantage that it implements a randomized descent in the space of locally optimal solutions. It is not clear, whether using a high value for k in the perturbation alone is sufficient to allow the algorithm to escape from bad solutions in combination with the

Better acceptance criterion. Our analysis of the run-time behavior of ILS in Section 4 shows that this is actually not the case. Other possibilities like allowing moves to worse solutions by the acceptance criterion have to be used to yield significantly improved performance.

3. Analysis of the QAP search space

3.1. Classes of QAP instances

It is known that the particular type of a QAP instance has a considerable influence on the performance of heuristic methods [48]. According to Taillard [48], most instances of QAPLIB we use in this article can be classified into four classes. These are unstructured, randomly generated instances in which the distance and flow matrix entries are integers randomly chosen from the interval [1, 100] (class i); instances with the distance matrix based on the Manhattan distance on a grid (class ii); real-life instances stemming from practical applications of the QAP [9,16, 28,41] (class iii); and instances which are randomly generated in a way that they resemble the structure of the real-life instances (class iv).

To differentiate among the classes of QAP instances, the flow dominance (*fd*) is commonly used. It is defined as the coefficient of variation of the flow matrix entries multiplied by 100:

$$fd(B) = 100 \cdot \frac{\sigma}{\mu}, \tag{3}$$

where

$$\mu = \frac{1}{n^2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} \quad \text{and}$$

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

A high flow dominance indicates that a large part of the overall flow is exchanged among relatively few items. Hence, class i instances have a rather low flow dominance whereas real-life problems, in general, have a large flow dominance. A disadvantage of the flow dominance is that it captures only the structure of one of two matrices, neglecting that of the distance matrix. Therefore, analogously to the flow dominance, we use the *distance dominance* (*dd*).

In general, real life problems often have many zero entries and the sparsity of the matrix can give an additional indication of the instance type. Let n_0 be the number of "0" entries in a matrix, then we define its sparsity *sp* as $sp = n_0/n^2$.

In Tables 1 and 2 are given the flow and the distance dominance and the sparsity of the sparser of the two matrices (typically, at most one matrix of these QAP instances has a sparsity larger than 0.1) for some instances of QAPLIB, ordered according to the four instance classes (the other table entries are explained in the following). In general, class iii and iv instances have the highest dominance values; the instances of class ii still have significantly larger flow dominance than the unstructured, randomly generated instances of class i.

3.2. Search space analysis of the QAP

Central to the search space analysis of combinatorial optimization problems is the notion of *search landscape* [40,50]. Intuitively, the search landscape can be imagined as a (multi-dimen-

Table 1
Given are the instance identifier, the flow, the distance dominance and the sparsity of some QAPLIB instances (columns 1–4)

Instance	$dd(A)$	$fd(B)$	sp	N_{opt}	avg^{ls}_{d-opt}	avg^{ils}_{d-opt}	r_{ls}	r_{ils}
Unstructured, randomly generated instances, class i								
tai20a	67.02	64.90	0.015	1				
tai25a	61.81	64.29	0.016	1				
tai30a	58.00	63.21	0.013	1				
tai35a	61.64	61.57	0.010	1				
tai40a	63.10	60.23	0.009	1				
tai60a	61.41	60.86	0.011	1				
tai80a	59.22	60.38	0.009	1				
rou20	65.65	64.43	0.010	1				
Instances with grid-distances, class ii								
nug30	52.75	112.48	0.316	4				
tho30	59.25	137.86	0.484	4				
tho40	53.20	155.54	0.585	4				
sko42	51.96	108.48	0.292	4				
sko49	51.55	109.38	0.304	8				
sko56	51.46	110.53	0.305	4				
sko64	51.18	108.38	0.308	8				
sko72	51.14	107.13	0.299	4				
					18.78	18.62	0.065	0.088
					23.83	23.64	0.064	0.059

28.69	28.32	0.100	0.208
33.75	33.61	0.041	0.054
38.86	38.81	0.020	0.107
58.82	58.71	0.025	0.006
78.90	78.77	0.022	0.049
18.50	18.10	0.124	0.114
25.93	23.81	0.262	0.406
26.27	24.86	0.328	0.472
36.11	35.21	0.194	0.273
37.96	35.18	0.302	0.499
44.58	43.40	0.213	0.234
51.62	49.51	0.254	0.448
58.88	56.33	0.303	0.353
67.38	65.31	0.264	0.284

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 3.1. The remaining entries give summary results of an analysis of the fitness–distance correlation of the QAP search space (see Section 3.2). 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 solutions returned by local search and iterated local search executed for n iterations, respectively, and r_{ls} and r_{ils} are the respective fitness–distance correlation coefficients.

sional) mountainous region with hills, craters, and valleys and the performance of stochastic local search methods strongly 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 search landscape is defined by

1. the set of all possible solutions $\mathcal{S}$;
2. an objective function that assigns to every $s \in \mathcal{S}$ a numerical value $f(s)$;
3. a distance measure $d(s, s')$ that gives the distance between solution s and s'.

An ILS algorithm follows a discontinuous trajectory in such a search landscape if we consider the search landscape that is defined by the neighborhood graph with respect to the iterative improvement method. For the global ILS guiding mechanism to be effective the characteristics of the search landscape topology like the relative location of locally optimal solutions, the average distance between local optima and the relative location of locally optimal solutions with respect to global optima are important. In particular the study of the correlation between cost and the distance to global optima (or best known solutions if global optima are not available) has proved to be a useful tool to judge the suitability of a search landscape for adaptive multi-start algorithms like ILS [7,18,27, 36]. This correlation is captured by the fitness–distance correlation (FDC) coefficient [27], which measures the linear correlation of the solution cost to the distance to the closest global optimum. Given a set of pairs $CD = \{(c_1, d_1), \ldots, (c_m, d_m)\}$ of cost values and distances, the correlation coefficient is defined as

$$r(C,D) = \frac{c_{CD}}{s_C \cdot s_D}, \tag{4}$$

where

$$c_{CD} = \frac{1}{m} \sum_{i=1}^{m} (c_i - \bar{c})(d_i - \bar{d}), \tag{5}$$

and $\bar{c}$, $\bar{d}$ are the average cost and the average distance, and s_C and s_D are the standard deviations of the costs and distances, respectively.

Table 2
Results of a search space analysis of the QAP search space

Instance	$dd(A)$	$fd(B)$	sp	N_{opt}	avg^{ls}_{d-opt}	avg^{ils}_{d-opt}	r_{ls}	r_{ils}
Real-life instances, class iii								
bur26a	15.09	274.95	0.223	96				
bur26b	15.91	274.95	0.223	690				
bur26c	15.09	228.40	0.257	96				

bur26d	15.91	228.40	0.257	790
bur26e	15.09	254.00	0.312	96
bur26g	15.09	279.89	0.211	96
chr25a	424.27	57.97	0.883	2
els19	52.10	531.02	0.637	1
kra30a	49.22	149.98	0.6	256
kra30b	49.99	149.98	0.6	128
ste36a	55.65	400.30	0.707	8
ste36b	100.79	400.30	0.707	8
Real-life like instances, class iv				
tai20b	128.25	333.23	0.410	1
tai25b	87.02	310.40	0.387	1
tai30b	85.20	323.91	0.432	1
tai35b	78.66	309.62	0.524	1
tai40b	66.75	317.22	0.503	1
tai50b	73.44	313.91	0.548	1
tai60b	76.83	317.82	0.548	1
tai80b	64.05	323.17	0.552	1
tai100b	80.42	321.34	0.552	1

21.12	20.15	0.027	0.457
21.26	19.72	0.021	0.678
22.31	15.39	0.569	0.867
20.29	18.19	0.471	0.787
18.43	14.91	0.479	0.853
18.47	13.89	0.666	0.876
22.92	21.71	0.252	0.359

16.85	13.76	0.550	0.654
25.23	24.10	0.251	0.413
24.83	23.25	0.312	0.379
30.98	28.37	0.295	0.504
29.59	24.76	0.381	0.778
17.32	14.69	0.420	0.576
21.65	23.83	0.456	0.703
27.71	25.47	0.264	0.518
32.04	30.17	0.328	0.525
37.76	35.39	0.329	0.626
47.94	45.39	0.156	0.363
56.88	52.28	0.366	0.540
77.47	75.56	0.150	0.457
95.23	92.34	0.546	0.608

See Table 1 for an explanation of the entries.

Here, we analyze the fitness–distance correlation for QAP instances. As the distance between solutions we measure the number of items which are placed on distinct locations in two solutions ϕ and ϕ', i.e., $d(\phi, \phi') = |\{i \,|\, \phi_i \neq \phi'_i\}|$. This is a direct extension of the well-known Hamming distance for bit strings. In the FDC analysis, we measure the distance to a globally optimal solution if available. Where optimal solutions are not available, we measure the distance to the best known solution. These best known solutions are conjectured to be optimal for instances of class ii, iii, and iv with up to 80 items, because they are the best solutions found by several algorithms including our ILS variants presented later. We refer to such solutions as *pseudo-optimal*.

For the FDC analysis, we have to take into account that many QAP instances have multiple optimal solutions. For example, for instances with a distance matrix defined by the distance between positions on a grid (class ii), it is known that these optimal solutions are at the maximally possible distance from the other optimal solutions (due to symmetries in the distance matrix). Hence, as also suggested in [27], we measure the distance to the *closest global optimum*. Unfortunately, the exact number of global optima for these instances is not known. Therefore, we determined in preliminary runs of our ILS algorithm a (possibly large) number of pseudo-optimal solutions: For small instances with $n < 40$ we stopped searching for more pseudo-optimal solutions if in 1000 trials of our ILS algorithm, each trial of at least 500 iterations, we did not find any more a pseudo-optimal solution which differed from any previously found pseudo-optimal solution; only on the larger instances of class ii we searched for the expected number of optimal solutions (either four or eight depending on the grid dimension).[2] In this process,

[2] It should be noted that after our research, Hahn et al. verified that, for example, on instance `kra30a` we actually had identified all optimal solutions [20].

for the instances of classes i and iv we found in each case only one single pseudo-optimal solution. Therefore, we conjecture that these instances have unique optimal solutions. The number of pseudo-optimal solutions found for each instance is indicated by N_{opt} in Tables 1 and 2.

We run two experiments in the fitness–distance analysis: In a first experiment (denoted as `E-ls`) we generated 5000 local optima (identical solutions at distance 0 have been eliminated) with 2-opt and measured the distance to the closest pseudo-optimal solution; in a second series of experiments (denoted as `E-ils`) we run 1000 times the ILS algorithm for n iterations (indicated by ILS(n)), again eliminating identical solutions. This second series of experiments is motivated by the fact that ILS (and several other stochastic local search methods) typically deal with solutions that are of a much better average quality than local optima that are obtained after only one single local search starting from a random solution. Hence, it is of strong interest how is the fitness–distance relationship between high quality solutions such as those that are typically found by (short) ILS runs.

Tables 1 and 2 also give the average distances of the local minima to the closest global optimum in `E-ls` (avg^{ls}_{d-opt}) and `E-ils` (avg^{ils}_{d-opt}), and the empirical FDC coefficients found in `E-ls` and `E-ils` (r_{ls} and r_{ils}, respectively). Fig. 2 gives plots of the fitness versus the distance to the closest optimum for one instance of each problem class.

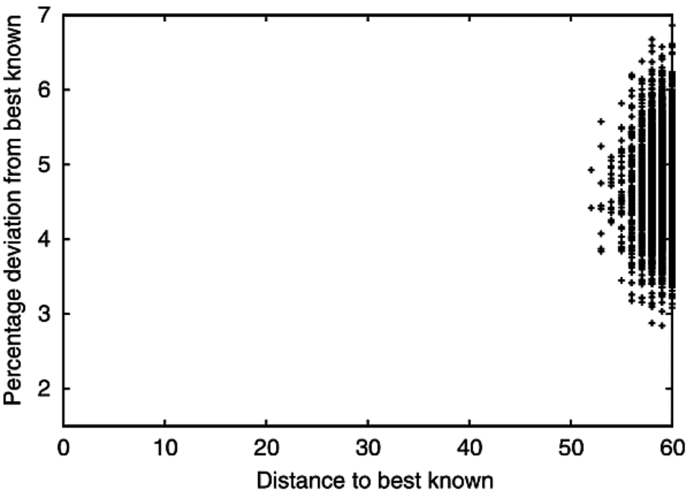
The fitness–distance analysis shows clear differences among the behavior for the different problem classes. For class i, all correlation coefficients are close to zero. Also the better solutions generated by ILS(n) do not show a significantly higher correlation. Regarding the average distances from global optima, 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, and the difference between avg^{ls}_{d-opt} and avg^{ils}_{d-opt} is minimal. Differently, for most instances of the other three classes, significant correlations exist. In fact, all correlations are statistically significant at the $\alpha = 0.05$ level with the only exception of r_{ls} for instances `bur26a` and `bur26b`. Comparing r_{ls} and r_{ils} for instances of classes ii to iv we find that r_{ils} is typically much larger than r_{ls}. It is also

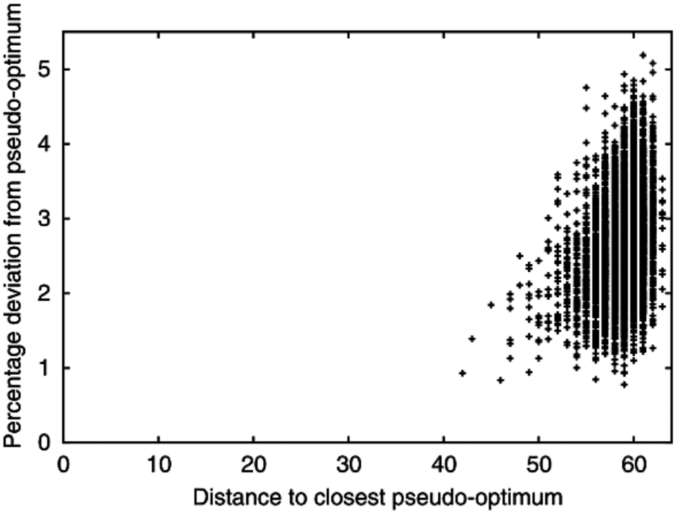
notable that for not too small instances avg^{ils}_{d-opt} is often smaller than avg^{ls}_{d-opt} by a factor of about 0.96. It is also in these two latter points where the instances of classes ii to iv show significant differences to those of class i. Comparing the instances of classes ii to iv we find that the correlation coefficients for instances of classes iii and iv are typically higher than those of class ii, which may indicate that the instances of these later two classes are easier for the ILS algorithms than those 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.[3]

In summary, we can conclude that—on average—the better the solution quality the closer a solution is to an optimal solution for the instances of classes ii to iv. These instances show a structure in the following sense: The more locations of items a solution has in common with an optimal solution, the better will be that solution on average. However, the FDC analysis also indicates that, when compared to the Traveling Salesman Problem, the local optima for the QAP are much more spread across the search space. This may also be an indication for the fact that QAP instances of a same dimensionality as TSP instances are much harder to solve.

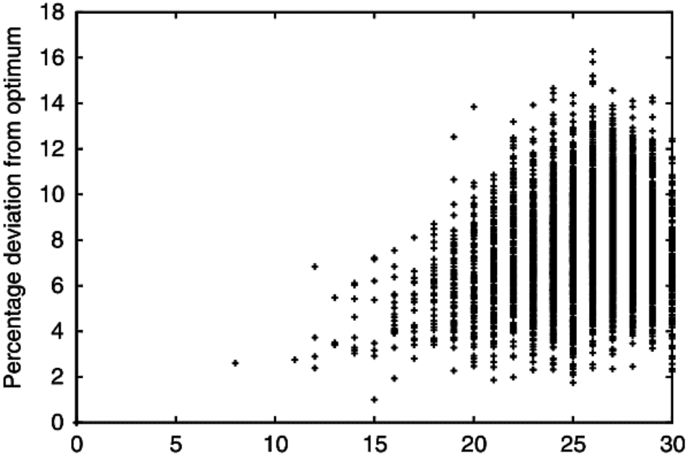
How can the observation on the fitness–distance correlations be exploited by an ILS algorithm? Note that the task of adaptive restart algorithms like ILS is to guide the search towards search space regions which contain very high quality solutions and, possibly, the global optimum. The most important guiding mechanism of these methods

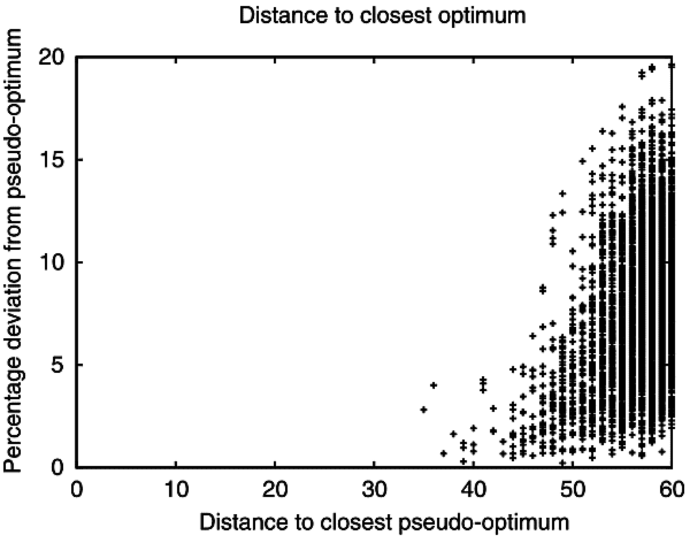
[3] An analysis of the FDC was also independently presented in [33]. However, the results of that FDC study were less conclusive than ours; this is probably due to the fact that the results in [33] do not consider, different from our study, that multiple optimal solutions exist at maximal distance for several instances and that distances should be measured to the closest global optimum. Additionally, the FDC analysis based on the short ILS runs for the QAP is done the first time here.


Percentage deviation from best known
7
6
5
4
3
2
0
10
20
30
40
50
60
Distance to best known

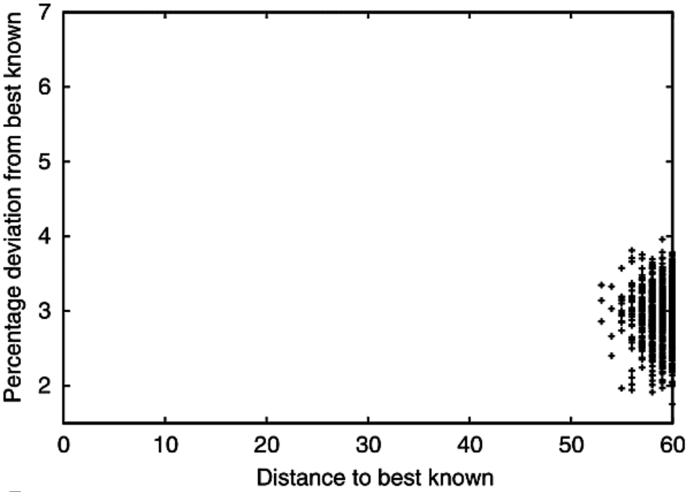


Percentage deviation from pseudo-optimum
Distance to closest pseudo-optimum
0
1
2
3
4
5
0
10
20
30
40
50
60

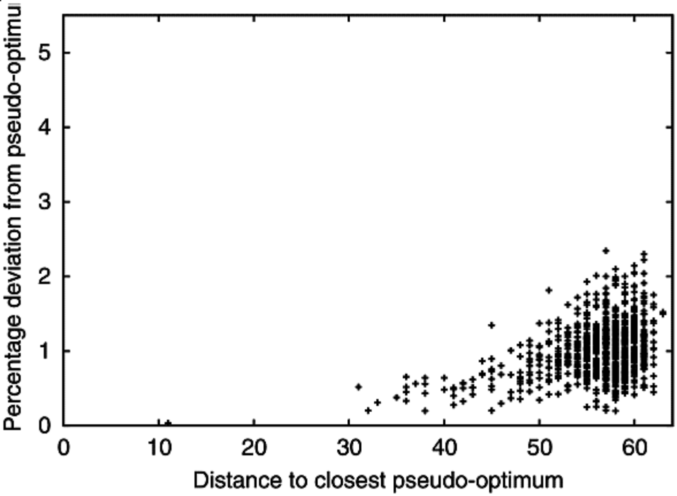

Percentage deviation from optimum
0
2
4
6
8
10
12
14
16
18
0
5
10
15
20
25
30

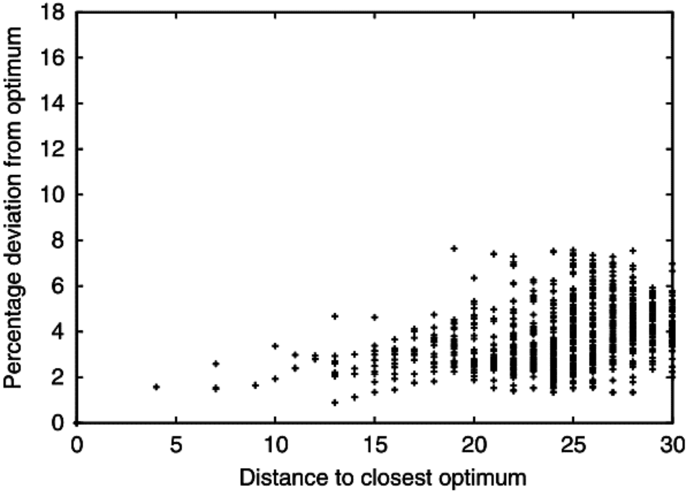


Distance to closest optimum
Percentage deviation from pseudo-optimum
Distance to closest pseudo-optimum
0
5
10
15
20
0
10
20
30
40
50
60


Percentage deviation from best known
7
6
5
4
3
2
0
10
20
30
40
50
60
Distance to best known

m


Percentage deviation from pseudo-optimu
0
1
2
3
4
5
0
10
20
30
40
50
60
Distance to closest pseudo-optimum


Percentage deviation from optimum
Distance to closest optimum
0
2
4
6
8
10
12
14
16
18
0
5
10
15
20
25
30

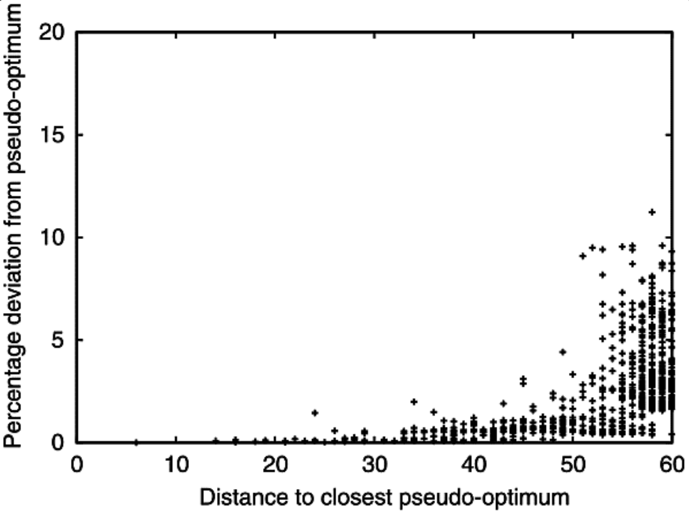


Fig. 2. Shown are the fitness–distance plots for four QAP instances, from top: `tai60a` (class i), `sko64` (class ii), `kra30a` (class iii), and `tai60b` (class iv). It is given for each instance on the left side a plot corresponding to 5000 2-opt solutions, on the right a plot based on 1000 solutions found by ILS(n). On the x-axis is given the distance to the closest pseudo-optimal solution (the x-axis ranges from the minimum to the maximum possible distance for each instance) and on the y-axis the percentage deviation from the best known or optimal solutions.

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 FDC describes the relation between the solution cost and the distance to the global optimum. 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 an intuitive explanation why it is reasonable that an ILS algorithm focuses strongly on the best solutions found so far.

4. Run-time behavior of ILS

In this section, we investigate the run-time behavior of the proposed ILS algorithm. ILS algorithms are randomized algorithms because they use random choices like random initial solutions and random mutations. For such algorithms, the time needed to reach a certain bound c on the solution cost is a random variable T_c with an associated distribution function $F(T_c)$. Knowledge on this distribution function can yield significant insights into the behavior of the algorithm and, therefore, we measure qualified run-time distributions along the lines proposed in [23,24,42,45].

4.1. *Empirical run-time distributions*

We empirically measure the distribution of T_c by running the ILS algorithm several times up to some maximal run-time t_{max} reporting each time a new improved solution is found the necessary run-time. For a given bound c on the solution cost, one can then determine a posteriori for each trial j the smallest time t_j required to find a solution within the given cost bound. The empirical RTD can be derived by first sorting the t_j values in nondecreasing order and then computing

$$\widehat{F}(T_c \leqslant t) = |\{j \mid t_{[j]} \leqslant t \wedge f(\phi_{[j]}) \leqslant c\}|/l, \qquad (6)$$

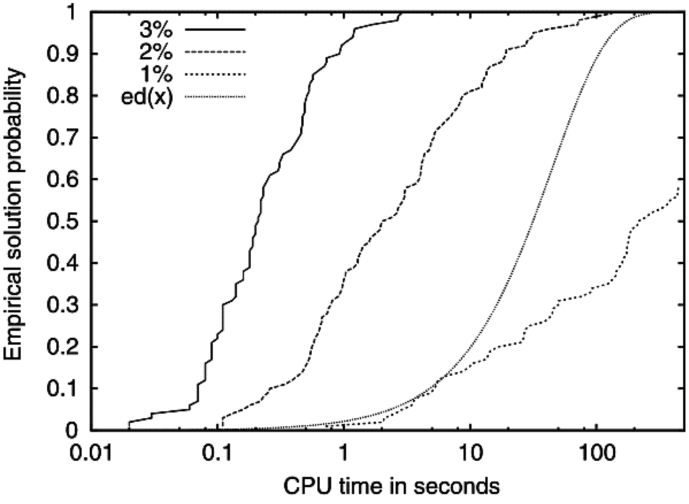
where $t_{[j]}$ is the run time of the *j*th longest trial and *l* is the total number of trials.

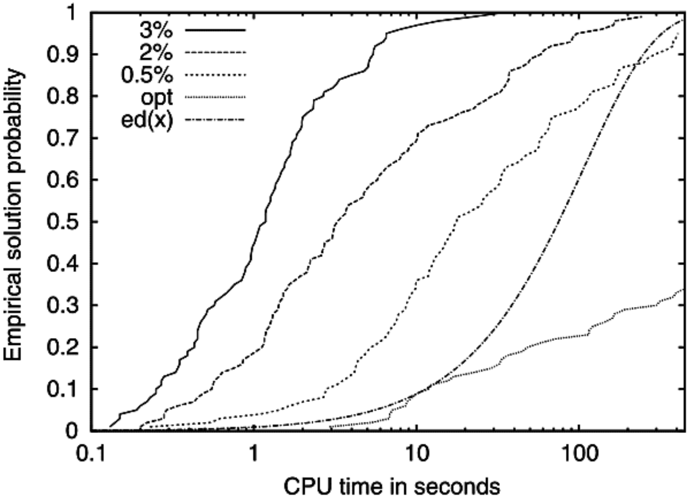
Fig. 3 gives empirical run-time distributions, measured across 100 independent trials, for one instance from each of the four classes with respect to various cost bounds given as the percentage excess over the pseudo-optimal solutions; the computation times refer to an UltraSparc II 167 MHz processor. The instances tested are `tai30a` (class i), `sko42` (class ii), `ste36a` (class iii), and `tai35b` (class iv). Additionally we give the cumulative distribution function of an exponential distribution (indicated by $ed(x)$) that approximates the lower part of one empirical distribution for each instance (the meaning of this curve is explained below).

The empirical RTDs show that the ILS algorithm is able to find the pseudo-optimal solutions with a high probability on three of the four instances. The only exception is instance `tai30a`, where only in one of 100 trials the pseudo-optimal solution was found. However, it is well known that for this type of instances optimal solutions are very hard to find [48].

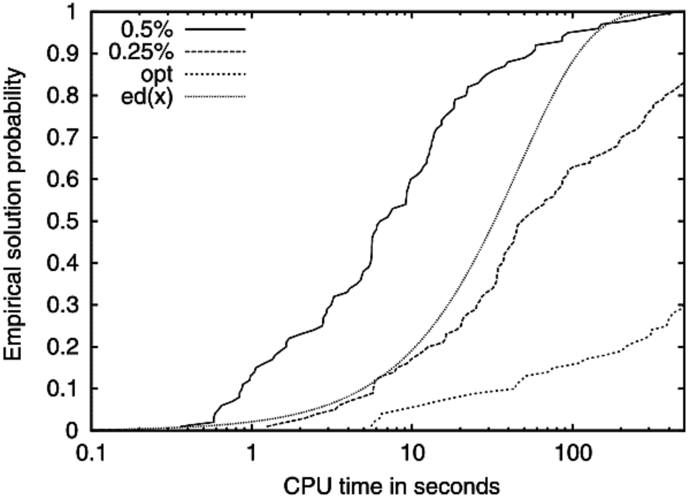
4.2. Stagnation behavior and exponential distributions

But what about the exponential distribution? Intuitively, a stochastic search algorithm shows stagnation behavior if for long trials, the probability of finding a solution that is at least as good as a given bound c can be improved by a random restart of the algorithm at some appropriately chosen cut-off time t_r. In fact, random restart is just an extreme form of diversification for an algorithm. Such stagnation behavior may easily be detected by comparing exponential distributions to the empirical RTDs. Applied to randomized algorithms, a well-known result from statistical theory about exponential distributions (memoryless property) [37] can be interpreted as follows: if for a given algorithm the random variable T_c is distributed exponentially, the probability of finding a solution when running the algorithm l times for time t is the very same as when running the algorithm once for time $l \cdot t$. Hence, to detect whether an algorithm shows stagnation behavior, we can compare the empirical RTDs against an





3%
2%
0.5%
opt
ed(x)
Empirical solution probability
CPU time in seconds
0.1
1
10
100
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



0.5%
0.25%
opt
ed(x)
Empirical solution probability
CPU time in seconds
0.1
1
10
100
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

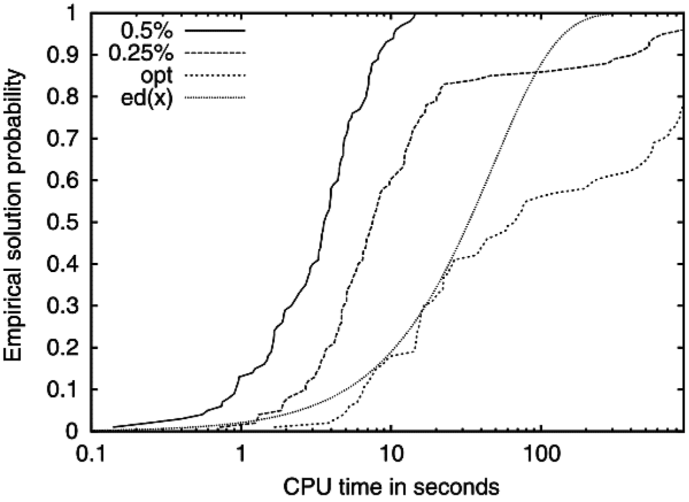


Fig. 3. The plots give empirical run-time distributions for QAP instances measured across 100 independent trials of an ILS algorithm with `Better` acceptance criterion w.r.t. several bounds on the required solution quality, given as the percentage deviation from the best known solution (the optimum is only known for `ste36a`). Given are the distributions for `tai30a` (upper left), `sko42` (upper right), `ste36a` (lower left) and `tai35b` (lower right).

exponential distribution. This leads to an easily applicable means of visually (as well as analytically; for details see [23]) detecting stagnation behavior. Intuitively, when using a logarithmic scale on the *x*-axis, this can be done by shifting an exponential distribution from the left to the empirical RTD until it first touches it; if from the tangential point on the empirical RTD develops (significantly) below the exponential curve, this indicates stagnation behavior. The same data can also be used to estimate optimal cutoffs for restarting the algorithm [23,45].

In Fig. 3, the exponential distributions were determined in the way described above and they indicate that the empirical RTDs for obtaining high quality solutions fall below this "idealized" exponential RTD. Therefore, by an appropriately determined cutoff time we easily can increase the probability of finding high quality solutions in the long run. This potentially large improvement is best illustrated with an example. When trying to solve instance `tai35b` to pseudo-optimality, consider an cutoff time of $t_r = 26$, with $\widehat{F}_{26} = 0.4$. If the ILS is restart with this cutoff time ten times, one can easily compute that the resulting, estimated solution probability would be 0.994, while the empirical RTD of the ILS algorithm without restarts reaches only a solution probability of 0.60 after $10 \cdot t_r = 260$ seconds. The resulting empirical RTD of such an ILS algorithm with restarts, given in Fig. 4, confirms the possible impressive improvements of ILS behavior.

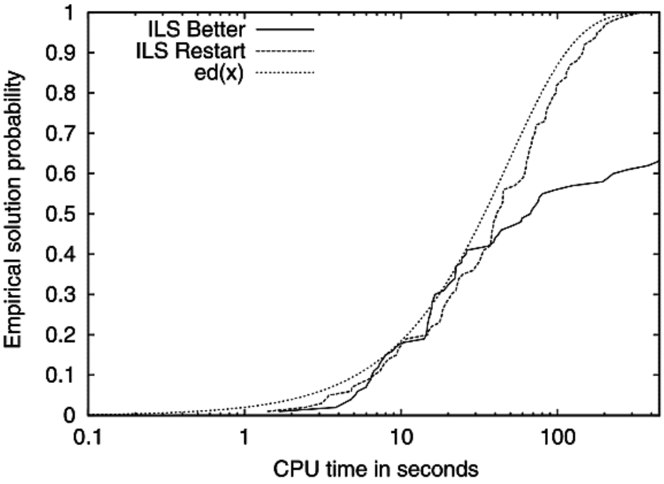


Fig. 4. Run-time distributions using an iterated `2-opt` algorithm that only accepts better solutions (ILS-better) and a restart version of ILS (ILS-restart) on instance `tai35b` for finding pseudo-optimal solutions. The restart algorithm uses a fixed cutoff of 26 seconds. Note that the run-time distribution of ILS-restart is well approximated by an exponential solution that is indicated by ed(x). The run-time distributions are based on 100 independent trials of each algorithm.

The reason for the possible improvements by restarting the ILS algorithm is that by accepting only better solutions, the ILS algorithm was, despite the fact that also rather large perturbations were allowed, not capable to efficiently explore regions of the search space which are at a larger distance from the current solution. It is important to note that a restart with a fixed cutoff-time is only an extreme form for increasing the exploration of the search space. A main disadvantage of such a restart is that (i) good solutions found in the past are not exploited and (ii) fixed cutoff times are not useful because good settings for the cutoff time are not known a priori. In the following we therefore propose several means of alleviating these two problems by exploring two main possibilities, namely (i) to exploit other types of acceptance criteria and (ii) to propose population-based ILS extensions, where the solutions in the population can explore different regions of the search space.

4.3. ILS with extended acceptance criteria

Instead of applying fixed cutoffs for restarting an ILS algorithm, a better idea is to make the decision to restart depend on the search progress by applying *soft* restarts. This suggests to model restarts by an acceptance criterion *Restart*(s, s'', *history*) that returns a random, new solution if the ILS algorithm does not find an improved solution for a fixed number of it_0 iterations; the assumption of such a restart is that the algorithm is stuck and that further progress is difficult. Such a soft restart is a very simple use of the search history in ILS acceptance criteria through the parameter it_0.

One possibility to favor strongly exploration of the search space is to accept s'' as the new solution irrespective of its cost, resulting in a random walk over locally optimal solutions. We call this acceptance criterion *RandomWalk*(s, s''). While *RandomWalk*(s, s'') does not exploit good solutions, an interpolation between the *RandomWalk* and *Better* acceptance criteria can be achieved by accepting worse solutions only with a certain probability. One such example is a simulated annealing type criterion. We denote this acceptance criterion by *LSMC*(s, s''), reminiscent of the fact that one of the first ILS algorithms, called *large step Markov chains* [32], used this acceptance criterion. In particular, s'' is accepted with a probability p given by

$$p(T, s, s'') = \begin{cases} 1 & \text{if } f(s'') < f(s), \\ \exp\left(\frac{f(s)-f(s'')}{T}\right) & \text{otherwise.} \end{cases} \tag{7}$$

T is a parameter called temperature and it is lowered during the run of the algorithm according to a temperature schedule like in Simulated Annealing.

4.4. Population-based ILS extensions

Maintaining a population of solutions provides an alternative way of augmenting exploration of the search space. ILS can easily be extended to a population-based algorithm in which each single solution follows a standard ILS algorithm [22,42]. In the simplest case, no interaction among the solutions takes place. When run with μ solutions for time t, such an approach corresponds to the sequential execution of μ ILS trials each for time t/μ. Certainly, variants that allow interaction among the solutions are more interesting; two such variants are described next.

Replace-worst. This variant starts with μ solutions each of which follows a standard ILS algorithm, except that every r iterations a copy of the current best solution replaces the worst solution in the population. This scheme gradually shifts the focus of the search towards the best solutions of the population and the parameter r determines how fast this shift takes place: If r is small, the search concentrates rapidly around possibly suboptimal solutions, if r is large, replace-worst behaves similar to multiple independent trials of an ILS algorithm.

Evolution strategies. This is a population-based extension that replaces the current population by a mechanism adapted from the $(\mu + \lambda)$-selection scheme from evolution strategies. In this scheme, the population comprises μ solutions and in each iteration λ new solutions are generated; the new set of solutions is then taken to comprise the μ best out of the $\mu + \lambda$ possible solutions. Since this way of population manipulation very strongly favors the best solutions, care has to be taken that the population does not converge too early. Therefore, we apply the following modification of the $(\mu + \lambda)$-selection. First, we set $\lambda = \mu$ and select each of the solutions in the population deterministically to undergo one iteration of the standard ILS algorithm. Then, the resulting $\mu + \lambda = 2 \cdot \mu$ solutions are sorted according to their cost and solutions are considered for insertion in the population in that order. A solution is inserted into the new population if the distance to any of solutions already member of the new population is larger than some minimum distance d_{min}. One main point in the algorithm is that d_{min} varies dynamically by subsequently lowering the actual value of d_{min} at run-time. 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, while allowing the algorithm to converge to high-quality solutions later on.

It is clear that these population-based ILS extensions share similarities to other population-based search metaphors like evolutionary algorithms and, in particular, to memetic algorithms [34]. In memetic algorithms, solutions are modified by mutation operators applied to single solutions and by crossover operators that exchange information between solutions and subsequently improved by a local search. A deliberate difference between the proposed population-based ILS extensions to memetic algorithms is that in ILS solutions are only modified by applications of a perturbation, which corresponds to mutations in the context of evolutionary algorithms.

5. Experimental results

In this section, we experimentally evaluate the performance of the proposed ILS algorithms on a wide range of QAP instances from QAPLIB and other sources. In the following, we refer to the ILS algorithm as described in Section 2 as `Better`, and to the ILS algorithms with acceptance criteria *LSMC*(s, s''), *RandomWalk*(s, s''), and *Restart*$(s, s'', history)$ as `LSMC`, `RW`, and

`Restart`, respectively. To the population-based extensions we refer to as `RepWorst` and `ES`, respectively.

This section reports the results for two series of experiments. In the first one, we compare the performance of the various ILS variants, robust tabu search [47] (`Ro-TS`), and $\mathscr{MAX}$–$\mathscr{MIN}$ Ant System ($\mathscr{MM}$AS) [44]. The latter two algorithms have been chosen since `Ro-TS` is known to perform very well on problem classes i and ii [19,48], $\mathscr{MM}$AS has been shown to be among the best available algorithms for structured instances as they occur in classes iii and iv [44], and all algorithms use the same underlying local search implementation. All algorithms are run for 10 independent trials and are given the same computation time (indicated by t_{max} in Tables 3 and 4; t_{max} corresponds to the time required by our `Ro-TS` implementation to do $1000 \cdot n$ iterations). The times are given as seconds on a SUN UltraSparc II machine with two UltraSparc I processors (167 MHz) with 0.5 MB external cache and 256 MB main memory. Only one single processor has been used due to the sequential implementation of the algorithms. In a second series of experiments, we give performance results for a high performing ILS variant on a large set of QAP instances from QAPLIB and new benchmark instances by Taillard [15]. Based on these results, we compare the performance of our algorithm to recent, high performing algorithms from the literature.

5.1. Parameter settings

The parameters for the ILS variants were set as follows. In `Better` and `RW` we set $k_{min} = 3$ and $k_{max} = 0.9 \cdot n$. 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 $it_0 = 2.5 \cdot k_{max}$. For `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 $T_{init} = 0.025 \cdot f(\phi')$, that is, a solution 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 $T_{i+1} = 0.9 \cdot T_i$. If in 100 iterations less than three times a worse solution was 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 and a setting of $k_{max} = 10$ for both, RepWorst and ES. In RepWorst, the first 30 iterations of the ILS runs are completely independent of each other and after iteration 30 we set $r = 3$. 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 15 or for 30 iterations no improved solution was found. The diversification consists in applying four iterations of RW with $k = n/2$ to each solution. Additionally, in the variants LSMC, RepWorst, and ES we initially set $k_{min} = k_{max}$ and from then on $k_{min} = \max\{3, k_{max} - it\}$. When the temperature is reset to T_{init}

in LSMC or diversification takes place in the population-based extensions, we also reset k_{min} as described before (by resetting also *it* to zero).

The parameter setting for Ro-TS and $\mathcal{MM}$AS are as proposed in the original articles [44,47].

5.2. Comparison of ILS variants

The computational results of the first series of experiments, where we compare the performance of the ILS variants, Ro-TS and $\mathcal{MM}$AS, are given in Table 3 for the instances of classes i and ii and in Table 4 for the instances of classes iii and iv. A first conclusion from the computational results is that, in general, with the proposed ILS 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 by how much an ILS extension improves over the performance of Better strongly depends on the instance class.

For the instances of classes i and ii (see Table 3) all extensions show a significantly improved performance over `Better`, the best performing ones being `ES` and `LSMC`, closely followed by `RepWorst` and `RW`. On the instances of classes iii and iv, the relative performance of the ILS algorithms is different. Here, `ES`, `RepWorst`, and `Restart` perform best. On most of these latter instances `Better` performs similar or slightly better than `RW`, while `RW` gives significantly higher quality solutions, on average, than `Better` on instances of classes i and ii. This fact can be explained to some extent with the different structure of these instances. 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 latter type of instances a bias towards the best solutions found during the search as in `Better` appears to be desirable.

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 slightly smaller than those of class iii and iv). Part of this success may be explained by the much larger number of local searches that can be applied in the same computation time by RW compared to

Table 3
Experimental results for ILS variants, Ro-TS and $\mathcal{MM}$AS for QAP instances from QAPLIB

Problem instance	Better	LSMC	RW	Restart	RepWorst	ES	Ro-TS	$\mathcal{MM}$AS	t_{max}
Random problems with entries uniformly distributed, class i									
tai20a			0.723		0.503		0.542		0.467
tai25a			1.181		0.876		0.896		0.823
tai30a			1.304		0.808		0.989		1.141
tai35a			1.731		1.110		1.113		1.371
tai40a			2.036		1.319		1.490		1.491
tai50a			2.127		1.496		1.491		1.968
tai60a			2.200		1.498		1.692		2.081
tai80a			1.775		1.198		1.200		1.576
Random flows on grids, class ii									
nug30			0.219		0.020		0.052		0.020
sko42			0.269		0.010		0.010		0.161
sko49			0.226		0.133		0.133		0.139
sko56			0.418		0.087		0.087		0.153
sko64			0.413		0.068		0.068		0.202
sko72			0.383		0.134		0.134		0.294
sko81			0.586		0.101		0.100		0.194
sko90			0.576		0.131		0.187		0.322
sko100a			0.358		0.115		0.161		0.257

0.500 0.344 **0.108** 0.428 9

0.869	0.656	**0.274**	1.751	17
0.707	0.668	**0.426**	1.286	30
1.010	0.901	**0.589**	1.568	51
1.305	1.082	**0.990**	1.131	75
1.574	1.211	**1.125**	1.900	150
1.622	1.349	**1.203**	2.484	265
1.219	1.029	**0.900**	2.103	670
0.013	**0.007**	0.013	0.042	30
0.002	**0.0**	0.025	0.104	92
0.090	**0.068**	0.076	0.150	135
0.102	**0.071**	0.088	0.118	211
0.079	**0.057**	0.071	0.243	308
0.139	**0.085**	0.146	0.243	455
0.100	**0.082**	0.136	0.223	656
0.262	**0.128**	**0.128**	0.288	895
0.191	**0.109**	0.128	0.191	1240

Given is the percentage deviation from the best known solutions over 10 independent trials of each algorithm. See the text for a description of the variants used.

Table 4
Experimental results for ILS and several extensions on QAP instances taken from QAPLIB for structured instances

Problem instance	Better	LSMC	RW	Restart	RepWorst	ES	Ro-TS	$\mathcal{MM}$AS	t_{max}
Real life instances, class iii									
bur26a-h	**0.0**	0.001	0.001	**0.0**					
kra30a	0.672	0.090	**0.0**	0.134					
kra30b	0.094	0.026	0.046	0.051					
ste36a	0.377	0.099	0.451	0.227					

ste36b	**0.0**	**0.0**	**0.0**	**0.0**
Randomly generated real-life like instances, class iv				
tai20b	0.045	**0.0**	0.045	**0.0**
tai25b	**0.0**	**0.0**	0.007	**0.0**
tai30b	**0.0**	**0.0**	0.093	**0.0**
tai35b	0.131	0.049	0.081	**0.0**
tai40b	**0.0**	**0.0**	0.204	**0.0**
tai50b	0.203	0.185	0.282	0.028
tai60b	0.029	0.059	0.645	0.023
tai80b	0.785	0.256	0.703	0.260
tai100b	0.219	0.096	0.711	0.202

0.0	**0.0**	0.002	**0.0**	44
0.0	**0.0**	0.268	0.314	30
0.031	**0.008**	0.023	0.049	30
0.071	**0.015**	0.155	0.181	54
0.0	**0.0**	0.081	**0.0**	54
0.0	**0.0**	**0.0**	**0.0**	10
0.0	**0.0**	**0.0**	**0.0**	41
0.0	**0.0**	0.107	**0.0**	73
0.0	**0.0**	0.064	**0.0**	117
0.0	**0.0**	0.531	**0.0**	177
0.042	0.033	0.342	**0.002**	348
0.005	**0.0**	0.417	0.005	633
0.222	0.383	1.031	**0.096**	1510
0.113	**0.083**	0.512	0.142	2910

See Table 3 for a description of the entries.

Better. For example, in the given computation time with RW on instance tai80b approximately 5000 local searches can be run, while Better can only apply about 1600 local searches. This difference is mainly due to the fact that in RW the value of k, which determines the strength of the perturbations, is always very small and, also due to the use of don't look bits described in Section 2, a local search after a small perturbation is much faster than after larger perturbations. To illustrate

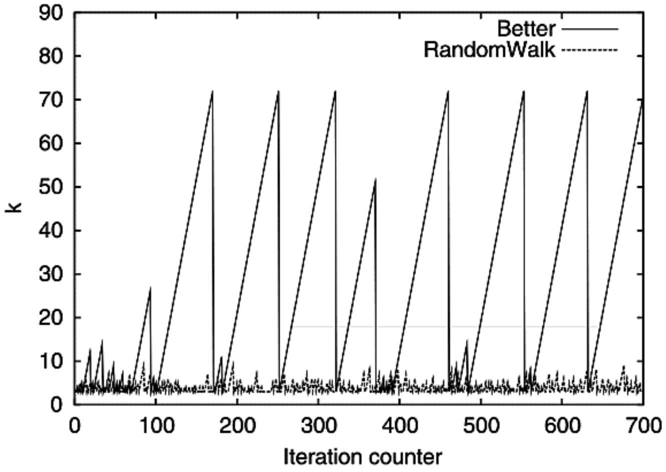


Fig. 5. Development of the perturbation strength for `Better` and `RW`.

the values of k used, in Fig. 5 we give the development of k versus the iteration counter for `Better` and `RW` on instance `tai80b`. In Fig. 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 $k = 51$; recall that after an improved solution is found, k is again set to k_{min}). In fact, if k_{max} is chosen too low in `Better`, its performance on many instances decreases strongly.

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`. On class iii and iv, the best four ILS 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, closely followed by the two population-based extensions and `Restart`.

Overall, the `ES` variant appears to perform particularly well. In fact, except for the instances of class i, `ES` typically has better average costs than the better between `Ro-TS` and $\mathcal{MM}$AS; the only further exceptions are the instances `tai50b` and `tai80b`. Taking this fact into account, from the computational results presented in [43] one can also conclude that for many of the larger instances from classes ii to iv, ES is superior to the genetic hybrids by Fleurent and Ferland [17] and hybrid ant system [19].[4] The very good performance of ES was also confirmed by the computational tests done within the Metaheuristics Network, a research and training network financed by the European Commission (see www.metaheuristics.net). In an extensive experimental analysis of metaheuristics (or, say, general-purpose stochastic local search methods) for the QAP that included ILS, simulated annealing, tabu search, memetic algorithms, and ant colony optimization algorithms, a slight variant of ES, which is presented in the next section, was found to be the overall best performing algorithm.

In summary, the comparison of the ILS algorithms to other algorithms that were tested in same experimental conditions shows that, despite their conceptual simplicity, they belong to the best performing algorithms for QAP instances with relatively high fitness–distance correlations as they occur in classes ii to iv.

5.3. Detailed experiments

In this section, we present additional results with a variant of ES that was tested in the experimental part of the Metaheuristics Network. We call this variant ES-MN; it essentially is a reimplementation of ES that (i) uses a more efficient local search implementation especially for asymmetric instances, based on observations in [35] on the transformation of asymmetric instances into symmetric ones, and (ii) slightly different parameter settings—ES-MN uses a population size of 25 and a setting of $k_{max} = 25$. The experiments reported in the following were run on a single 1.2 GHz Athlon MP processor running under SuSe Linux 7.3 with 1 GB of RAM. We compare the results of these experiments to two algorithms, the memetic algorithm by Merz and Freisleben [33] and the memetic algorithm by Drezner [14].[5] Both memetic

[4] Note that reactive tabu search [4] performs better than Ro-TS on the class i instances, but typically worse than Ro-TS on the other classes [42]. Hence, ES is also superior to reactive tabu search on instances of classes ii to iv.

[5] Note that the memetic algorithm of Merz and Freisleben

also uses the local search speed-up techniques based on transforming asymmetric into symmetric instances; the memetic algorithm of Drezner was only applied to symmetric instances.

algorithms were shown to perform better than several competitors on a number of instances and are recent state-of-the-art algorithms for the QAP.

We run ES-MN on QAPLIB instances with 25 or more items. For many of these instances, ES-MN found in every single trial the optimal or pseudo-optimal solutions in modest computation times. The required computation times for these instances are indicated in Table 5. We did not include instances in this table, where the average computation times were below 0.5 seconds; this was the case for all bur26X and esc32X in-

Table 5

stances, as well as for the instances esc64a, nug25, nug27, nug28, and tai25b.

Only for very few stochastic local search algorithms for the QAP it has been reported that they were able to obtain with a very high probability the optimum or pseudo-optimal solutions in each trial for large instances with up to 100 items. The only one we are aware of is the memetic algorithm by Merz and Freisleben (MA-MF) [33]. MA-MF was run on nine QAPLIB instances on a Pentium II 300 MHz computer, a machine which is roughly 4.05 times slower than our machine (comparison

Statistics on the computation times (given in seconds) for solving QAPLIB instances with ES-MN to optimality or their best known solutions, if optimal ones are not available

Instance	Optimum	Average cost	t_{min}	t_{avg}	t_{max}
chr25a		3796			3796.0
escl28		64			64.0
kra30a		88900			88900.0
kra30b		91420			91420.0
kra32		88700			88700.0
nug30		6124			6124.0
lipa40a		31538			31538.0
lipa50a		62093			62093.0
lipa60a		107218			107218.0
lipa70a[†]		169755			169755.0
lipa60b		2520135			2520135.0
lipa70b[†]		4603200			4603200.0
lipa80b[†]		7783962			7783962.0
lipa90b[†]		12490441			12490441.0
ste36a		9526			9526.0
ste36b		8653			8653.0
ste36c		8239110			8239110.0
sko42		15812			15812.0
sko49		23386			23386.0

sko56	34458	34458.0
sko64	48498	48498.0
tai25a	1167256	1167256.0
tai30a	1818146	1818146.0
tai35a[†]	2422002	2422002.0
tai30b	637117113	637117113.0
tai35b	283315445	283315445.0
tai40b	637250948	637250948.0
tai50b	458821517	458821517.0
tai60b	608215054	608215054.0
tai80b[†]	818415043	818415043.0
tai100b[†]	1185996137	1185996137.0
tho30	149936	149936.0
tho40[†]	240516	240516.0
wil50	48816	48816.0

0.25	1.42	7.94
0.09	1.67	5.88
0.15	0.63	3.55
0.13	1.38	6.55
0.19	0.61	2.24
0.14	1.29	6.52
0.58	2.14	6.62
1.37	7.32	45.65
3.86	57.24	279.70

14.27	57.24	451.23
0.31	2.84	21.17
0.71	3.97	30.21
2.00	14.69	48.09
2.09	33.31	171.80
0.54	4.59	24.02
0.14	1.09	2.00
0.59	2.28	10.56
1.21	3.09	12.96
5.28	170.21	851.64
2.39	45.08	191.59
5.54	30.86	185.89
0.47	12.10	64.23
0.95	38.31	325.83
10.85	139.55	415.80
0.47	0.85	1.97
0.33	1.36	3.88
0.83	1.57	4.52
2.47	5.95	23.23
4.30	12.61	53.48
23.67	79.62	340.86
42.80	140.70	696.15

0.15	0.91	3.33
3.12	187.4	561.7
4.65	62.70	364.20

All averages refer to 100 trials per instance except of those marked by †, were 25 trials were run. The computation times given indicate the minimum time, the average time, and the maximum time in seconds necessary to find optimal or best known solutions.

Table 6
Average computation times (measured over 30 trials per instance) for the memetic algorithm of [33] to optimal or pseudo-optimal solutions on some QAPLIB instances

Instance	t_{avg}	Instance	t_{avg}	Instance	t_{avg}	Instance	t_{avg}
chr25a	2.6	bur26a	1.0	nug30	7.1	kra30a	2.7
ste36a	36.7	tai60b	23.2	tai80b	258.3	tai100b	629.1

Times are given in seconds for a Pentium II 300 MHz machine; averages are taken from [33].

via SPEC). The average times required by the memetic algorithm to find the pseudo-optimal solutions are given in Table 6. A comparison to the times taken by ES, using the correction factor of approximately 4, shows that both algorithms reach similar performance. The largest differences can be found on instances ste36a, where ES is roughly double as fast as the memetic algorithm, while on instance tai60b the memetic algorithm is roughly double as fast as ES. However, ES has the additional advantage of being conceptually more simple, since it does not use any recombination between solutions. In fact, it is widely acknowledged in the evolutionary computation community that finding good recombination operators is difficult.

On other instances of QAPLIB we performed experiments using a 1200 seconds upper limit on the computation time; the only exception are instances `tai150b` and `tho150`, which were the largest ones we tested with 150 items each, for which we allowed 3600 seconds. The computational results for these experiments are given in Table 7. The largest deviations from the best known solutions were observed for the instances of class i, which, however, are considered not to be of practical importance [48]. On the instances of class ii, `ES-MN` achieved excellent performance, finding in many trials the best known solutions and obtaining very low average deviations from the pseudo-optimal solutions. Also on the only remaining instance of class iv, instance `tai150b`, `ES-MN` achieved excellent performance and reached an average deviation from the best known solution of less than 0.1%.[6] Regarding the average performance on instances of classes ii and iv that

[6] All instances of class iii are solved to their optimal or

pseudo-optimal solutions in all the trials we run with ES-MN within a few seconds.

have been reported in the literature, to the best of our knowledge, ES-MN appears to be best; however, the computation times we allowed are larger than, e.g., in [33]. The only published results at the time of writing a revised version of this paper that are close to the ones presented here stem from a memetic algorithm by Drezner [14]. That memetic algorithm took computation times of around 34 minutes on a Pentium 600 MHz CPU (we estimate that 1200 seconds on our machine correspond roughly to 40 minutes on a Pentium 600 MHz CPU using again data from SPEC) for the instances with 100 items. Within the time limits we imposed, we obtain on most sko100X instances an average deviation from the best known solutions that is roughly halve the one found by Drezner's MA.[7]

Finally, we run ES-MN on new instances proposed by Taillard that were designed to be hard for stochastic local search algorithms. In Table 8 we present the computational results for the 20 instances of size 75; smaller instances of size 27 and 43 could all be solved to their known optimum or the pseudo-optimal solutions in fractions of a seconds for the size 27 instances and in a few seconds for the size 43 instances. Except for two instances, all the instances of size 75 could be solved to the best known solutions as reported in [15] in reasonable computation times. Hence, these instances appear not to be as hard as they were originally conceived. Additionally, these results confirm the

[7] To gather further evidence for the very good performance of `ES-MN`, we run `Ro-TS` on the `sko100X` instances for the same computation time limits as `ES` (`Ro-TS` can do roughly 1.5 million iterations within these time limits). `Ro-TS` was not able to get once the best known solutions on the `sko100X` instances and the average deviations from the best known solutions are typically by a factor of two to five larger than those obtained by `ES-MN`.

Table 7
Statistics on the quality and computation times (given in seconds) for QAPLIB instances that were not solved in all trials within the given computation time limits with `ES-MN`

Instance	Best known	% opt	% best	% average	% worst	t_{avg}	t_{max}
`tai40a`[†]	3139370		0.0	0.074			
`tai50a`	4941410		0.0	0.34			
`tai60a`	7205962		0.0	0.68			

tai80a	13540420	0.0	0.46
tai100a	21123042	0.0	0.54
sko72[†]	66256	0.88	0.0
sko81[†]	90998	0.52	0.0
sko90[†]	115534	0.40	0.0
sko100a	152002	0.30	0.0
sko100b	153890	0.50	0.0
sko100c	147862	0.60	0.0
sko100d	149576	0.0	0.0013
sko100e	149150	0.70	0.0
sko100f	149036	0.0	0.023
wil100	273038	0.1	0.0
tho150	8133398	0.0	0.041
tai150b	498896643	0.0	0.0003

0.28	0.43	721.2	1200
0.61	0.82	673.6	1200
0.82	0.95	446.5	1200
0.62	0.71	764.3	1200
0.69	0.78	587.8	1200
0.0012	0.018	399.8	1200
0.0074	0.013	575.9	1200
0.0057	0.024	635.5	1200
0.012	0.030	471.3	1200
0.0068	0.020	438.9	1200
0.0023	0.011	688.1	1200
0.021	0.064	710.2	1200
0.0013	0.0094	409.7	1200

0.037	0.068	699.7	1200
0.0041	0.0081	380.7	1200
0.068	0.095	2173.6	3600
0.095	0.21	2200.8	3600

Averages are given over 10 trials per instances; on some instances, which are indicated by †, 25 trials were run. % opt gives the percentage of pseudo-optimal solutions found and % best, % average, and % worst give the percentage deviation from the pseudo-optimum of the best, average and worst solution cost over all trials. t_{avg} is the average computation time when the best solution in a trial was reached.

Table 8
Statistics on the quality and the computation times (given in seconds) for solving new, hard instances from Taillard [15] with `ES-MN`

Instance	Best known	% opt	% best	% average	% worst	t_{avg}	t_{max}
`tai75e01`	14488	1.0	0.0				
`tai75e02`	14444	0.2	0.0				
`tai75e03`	14154	1.0	0.0				
`tai75e04`	13694	1.0	0.0				
`tai75e05`	12884	1.0	0.0				
`tai75e06`	12534	1.0	0.0				
`tai75e07`	13782	1.0	0.0				
`tai75e08`	13948	1.0	0.0				
`tai75e09`	12650	1.0	0.0				
`tai75e10`	14192	1.0	0.0				
`tai75e11`	15250	1.0	0.0				
`tai75e12`	12760	0.68	0.0				
`tai75e13`	13024	1.0	0.0				
`tai75e14`	12604	1.0	0.0				
`tai75e15`	14294	1.0	0.0				
`tai75e16`	14204	1.0	0.0				
`tai75e17`	13210	1.0	0.0				
`tai75e18`	13500	1.0	0.0				

tai75e19	12060	1.0	0.0
tai75e20	15260	1.0	0.0
0.0	0.0	108.7	1200
1.96	4.62	510.3	1200
0.0	0.0	47.2	1200
0.0	0.0	41.4	1200
0.0	0.0	99.3	1200
0.0	0.0	66.1	1200
0.0	0.0	336.7	1200
0.0	0.0	180.1	1200
0.0	0.0	38.9	1200
0.0	0.0	46.7	1200
0.0	0.0	33.1	1200
0.52	4.08	536.0	1200
0.0	0.0	91.6	1200
0.0	0.0	61.8	1200
0.0	0.0	33.5	1200
0.0	0.0	33.6	1200
0.0	0.0	46.0	1200
0.0	0.0	62.4	1200
0.0	0.0	44.5	1200
0.0	0.0	46.9	1200

Averages are given over 25 trials per instance. See Table 7 for a description of the entries.

good behavior of ES-MN when compared to the memetic algorithm by Drezner, which could solve only 11 of the 20 instances in each of the 20 trials per instance within a maximum computation time of about 37 minutes (that is, about 2220 seconds) on a Pentium 600 MHz CPU.

6. Conclusions

In this article, we have presented and analyzed iterated local search (ILS) algorithms for the quadratic assignment problem. In our research we started from a basic ILS algorithm.[8] A detailed analysis of the run-time behavior of this basic version revealed that its performance was compromised by a strong stagnation behavior. The insights gained from this analysis lead us to develop much more performing ILS algorithms including new population-based extensions of ILS. In fact, our computational results showed that the best performing ILS variants are new state-of-the-art algorithms for the QAP.

In summary, the contributions of this article are (i) the extensive analysis of the fitness–distance correlation on QAP instances, (ii) a detailed analysis of the run-time behavior of a basic ILS algorithm, (iii) the strong evidence provided that acceptance criteria play a crucial role in the success of ILS algorithms and therefore should be examined carefully when engineering an ILS algorithm, (iv) the development of a new, population-based variant of ILS for the QAP, and (v) a detailed experimental analysis of the best performing ILS variant, `ES-MN`, which established it as a state-of-the-art algorithm for the QAP.

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 direction to follow, would be to make a stronger use of the history component which is indicated in the general algorithmic outline of ILS given in Fig. 1. Such extensions could strongly benefit from the use of long-term memory techniques used in tabu search. Interestingly, the best performing ILS variants

[8] Independent from our ILS approach in [49] an ILS algorithm similar to the one presented in Section 2 has been

implemented. However, that article lacks an analysis of the algorithm and the algorithm of [49] is by far outperformed by our best ILS variants.

were actually population-based algorithms; it would be interesting to obtain general insights into why and when a population of solutions can give better performance than algorithms based on a single solution. An interesting research direction arises from the observation of the stagnation behavior observed in the run-time behavior of the ILS algorithm. In [3,47] 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 5, the required run-time to find the pseudo-optimal solution for unstructured instances of type i (defined in Section 3) 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 [42]. 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 instances being tackled lead to such a behavior.

Acknowledgments

Part of this work has been supported by a Madame Curie Fellowship awarded to Thomas Stützle (CEC-TMR Contract No. ERB4001-GT973400) and by the Metaheuristics Network, a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] K.M. Anstreicher, N.W. Brixius, J.-P. Goux, J. Linderoth, Solving large quadratic assignment problems on computa-

tional grids, Mathematical Programming 91 (3) (2002) 563–588.

[2] E. Balas, A. Vazacopoulos, Guided local search with shifting bottleneck for job shop scheduling, Management Science 44 (2) (1998) 262–275.

[3] R. Battiti, G. Tecchiolli, Parallel biased search for combinatorial optimization: Genetic algorithms and TABU, Microprocessor and Microsystems 16 (7) (1992) 351–367.

[4] R. Battiti, G. Tecchiolli, The reactive tabu search, ORSA Journal on Computing 6 (2) (1994) 126–140.

[5] E.B. Baum, Iterated descent: A better algorithm for local search in combinatorial optimization problems, Manuscript, 1986.

[6] J.L. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on Computing 4 (4) (1992) 387–411.

[7] K.D. Boese, Models for iterative global optimization, PhD thesis, University of California, Computer Science Department, Los Angeles, 1996.

[8] N.W. Brixius, K.M. Anstreicher, The Steinberg wiring problem, Technical Report, The University of Iowa, USA, October 2001.

[9] R.E. Burkard, J. Offermann, Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme, Zeitschrift für Operations Research 21 (1977) B121–B132.

[10] R.K. Congram, C.N. Potts, S. van de Velde, An iterated dynasearch algorithm for the singlemachine total weighted tardiness scheduling problem, INFORMS Journal on Computing 14 (1) (2002) 52–67.
[11] D.T. Connolly, An improved annealing scheme for the QAP, European Journal of Operational Research 46 (1990) 93–100.
[12] V.-D. Cung, T. Mautor, P. Michelon, A. Tavares, A scatter search based approach for the quadratic assignment problem, in: T. Baeck, Z. Michalewicz, X. Yao (Eds.), Proceedings of ICEC'97, IEEE Press, 1997, pp. 165–170.
[13] J.W. Dickey, J.W. Hopkins, Campus building arrangement using TOPAZ, Transportation Science 6 (1972) 59–68.
[14] Z. Drezner, A new genetic algorithm for the quadratic assignment problem, INFORMS Journal on Computing 15 (3) (2003) 320–330.
[15] Z. Drezner, P. Hahn, É.D. Taillard, A study of quadratic assignment problem instances that are difficult for meta-heuristic methods, Annals of Operations Research, in press.
[16] A.N. Elshafei, Hospital layout as a quadratic assignment problem, Operations Research Quarterly 28 (1977) 167–179.
[17] C. Fleurent, J.A. Ferland, Genetic hybrids for the quadratic assignment problem, in: P.M. Pardalos, H. Wolkowicz (Eds.), Quadratic Assignment and Related Problems, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 16, American Mathematical Society, 1994, pp. 173–187.

[18] C. Fonlupt, D. Robillard, P. Preux, E.-G. Talbi, Fitness landscapes and the performance of metaheuristics, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, MA, 1999, pp. 255–266.

[19] L.M. Gambardella, É.D. Taillard, M. Dorigo, Ant colonies for the quadratic assignment problem, Journal of the Operational Research Society 50 (2) (1999) 167–176.

[20] P.M. Hahn, W.L. Hightower, T.A. Johnson, M. Guignard-Spielberg, C. Roucairol, Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem, Yugoslavian Journal of Operational Research 11 (1) (2001).

[21] P. Hansen, N. Mladenović, An introduction to variable neighborhood search, in: S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, MA, 1999, pp. 433–458.

[22] I. Hong, A.B. Kahng, B.R. Moon, Improved large-step Markov chain variants for the symmetric TSP, Journal of Heuristics 3 (1) (1997) 63–81.

[23] H.H. Hoos, T. Stützle, Stochastic Local Search—Foundations and Applications, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[24] H.H. Hoos, T. Stützle, Evaluating las vegas algorithms—pitfalls and remedies, in: Proceedings of the Fourteenth

Conference on Uncertainty in Artificial Intelligence UAI-98, Morgan Kaufmann Publishers, San Francisco, CA, 1998, pp. 238–245.

[25] D.S. Johnson, L.A. McGeoch, Experimental analysis of heuristics for the STSP, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002, pp. 369–443.

[26] D.S. Johnson, L.A. McGeoch, The travelling salesman problem: A case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley, 1997, pp. 215–310.

[27] T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: L.J. Eshelman (Ed.), Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufman, 1995, pp. 184–192.

[28] J. Krarup, P.M. Pruzan, Computer-aided layout design, Mathematical Programming Study 9 (1978) 75–94.

[29] H.R. Lourenc, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 57, Kluwer Academic Publishers, Norwell, MA, 2002, pp. 321–353.

[30] V. Maniezzo, Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem, INFORMS Journal on Computing 11 (4) (1999) 358–369.

[31] O. Martin, S.W. Otto, Combining simulated annealing

with local search heuristics, Annals of Operations Research 63 (1996) 57–75.

[32] O. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the traveling salesman problem, Complex Systems 5 (3) (1991) 299–326.

[33] P. Merz, B. Freisleben, Fitness landscape analysis and memetic algorithms for the quadratic assignment problem, IEEE Transactions on Evolutionary Computation 4 (4) (2000) 337–352.

[34] P. Moscato, Memetic algorithms: A short introduction, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, UK, 1999, pp. 219–234.

[35] P. Pardalos, F. Rendl, H. Wolkowicz, The quadratic assignment problem: A survey and recent developments, in: P. Pardalos, H. Wolkowicz (Eds.), Quadratic Assignment and Related Problems, American Mathematical Society, Providence, RI, 1994, pp. 1–42.

[36] C.R. Reeves, Landscapes, operators and heuristic search, Annals of Operations Research 86 (1999) 473–490.

[37] V.K. Rohatgi, An Introduction to Probability Theory and Mathematical Statistics, John Wiley, 1976.

[38] S. Sahni, T. Gonzalez, P-complete approximation problems, Journal of the ACM 23 (3) (1976) 555–565.

[39] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on Computing 2

(1990) 33–45.

[40] P.F. Stadler, Towards a theory of landscapes, Technical Report SFI-95-03-030, Santa Fe Institute, 1995.

[41] L. Steinberg, The backboard wiring problem: A placement algorithm, SIAM Review 3 (1961) 37–50.

[42] T. Stützle, Local search algorithms for combinatorial problems—Analysis, improvements, and new applications, PhD thesis, FB Informatik, TU Darmstadt, 1998.

[43] T. Stützle, M. Dorigo, ACO algorithms for the quadratic assignment problem, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, 1999. Also available as Technical Report IRIDIA/99-02, Université de Bruxelles, Belgium, 1999.

[44] T. Stützle, H.H. Hoos, $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System, Future Generation Computer Systems 16 (8) (2000) 889–914.

[45] T. Stützle, H.H. Hoos, Analysing the run-time behaviour of iterated local search for the travelling salesman problem, in: P. Hansen, C. Ribeiro (Eds.), Essays and Surveys on Metaheuristics, Operations Research/Computer Science Interfaces Series, Kluwer Academic Publishers, Norwell, MA, 2001, pp. 589–611.

[46] T. Stützle, Iterated local search for the quadratic assignment problem, Technical Report AIDA-99-03, FG Intellektik, FB Informatik, TU Darmstadt, 1999.

[47] É.D. Taillard, Robust taboo search for the quadratic assignment problem, Parallel Computing 17 (1991) 443–455.

[48] É.D. Taillard, Comparison of iterative searches for the

quadratic assignment problem, Location Science 3 (1995) 87–105.
[49] É.D. Taillard, L.M. Gambardella, Adaptive memories for the quadratic assignment problem, Technical Report IDSIA-87-97, IDSIA, Lugano, Switzerland, 1997.
[50] E.D. Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference, Biological Cybernetics 63 (1990) 325–336.