The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms

Tommaso Schiavinotto and Thomas Stützle
Darmstadt University of Technology, Computer Science Department
Alexanderstr. 10, 64283 Darmstadt, Germany
{schiavin,tom}@intellektik.informatik.tu-darmstadt.de

Abstract

The linear ordering problem is an $\mathcal{NP}$-hard problem that arises in a variety of applications. Due to its interest in practice, it has received considerable attention and a variety of algorithmic approaches to its solution have been proposed. In this paper we give a detailed search space analysis of available LOP benchmark instance classes that have been used in various researches. The large fitness-distance correlations observed for many of these instances suggest that adaptive restart algorithms like iterated local search or memetic algorithms, which iteratively generate new starting solutions for a local search based on previous search experience, are promising candidates for obtaining high performing algorithms. We therefore experimentally compared two such algorithms and the final experimental results suggest that, in particular, the memetic algorithm is the new state-of-the-art approach to the LOP.

1 Introduction

Given an $n \times n$ matrix C, the linear ordering problem (LOP) is the problem of finding a permutation π of the column and row indices $\{1, ..., n\}$ such that the value

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{\pi(i)\pi(j)}$$

is maximized. In other words, the goal is to find a permutation of the columns and rows of matrix C such that the sum of the elements in the upper triangle is maximized.

The LOP arises in a large number of applications in such diverse fields as economy, sociology, graph theory, archaeology, and task scheduling [10]. Two well known examples of the LOP are the triangularization of input-output matrices of an economy, where the optimal ordering allows economists to extract some information about the stability of the economy or the stratification problem in archaeology, where the LOP is used to find the most probable chronological order of samples found in different sites. The matrix that describes the problem is known as Harris Matrix.

The LOP is $\mathcal{NP}$-hard, that is, we cannot expect a polynomial time algorithm for its solutions. However, the LOP arises in a variety of practical applications [10] and therefore algorithms for its efficient solution are required. Several exact and heuristic algorithms were proposed in the literature. Exact algorithms include a branch & bound algorithm that uses a LP-relaxation for the lower bound by Kaas [14], a branch & cut algorithm proposed by Grötschel, Jünger, and Reinelt [10] and a combined interior point / cutting plane algorithm by Mitchell and Borchers [22]. State-of-the-art exact algorithms can solve fairly large instances from specific instance classes with up to a few hundred columns and rows, while they fail on other instances of other classes of much smaller size. Independent of the type of instances solved, the computation time of exact algorithms increases strongly with instance size.

The LOP was also tackled by a number of heuristic algorithms. These include constructive algorithms like Becker's greedy algorithm [3], local search algorithms like the $\mathcal{CK}$ heuristic by Chanas and Kobylanski [8], as well as a number of metaheuristic approaches such as elite tabu search and scatter search, presented in a series of papers by Martí, Laguna, and Campos [17, 7, 6], or iterated local search (ILS) algorithms [9, 25]. In particular, ILS approaches appear currently to be the most successful metaheuristics, as judged by their performance on a number of available LOP benchmark instances [9, 25].

Available algorithms have typically been tested on a number of classes of real-world as well as randomly generated instances. However, few is known about how the performance of current state-of-the-art algorithms depends on specific characteristics of the various available LOP instance classes neither how differences among the instances translate into differences in their search space characteristics. First steps into answering these open questions were undertaken in [25].

The main contributions of this article are the following. First, we give a detailed analysis of the search space characteristics of all the instance classes introduced in the major algorithmic contributions to the LOP. This includes a structural analysis, where standard statistical information is gathered as well as an analysis of the main search space characteristics such as autocorrelation [27, 33] and fitness-distance analysis [13]. A second contribution is the detailed analysis of two metaheuristics, an iterated local search algorithm [18] and a memetic algorithm [23]. As we will see, their relative performance depends on the particular LOP instance class to which they are applied. Interestingly, a detailed analysis shows that there is some correlation between specific search space characteristics and the hardness of the instances as encountered by the two metaheuristics. Computational comparisons of the two algorithms to known metaheuristic approaches establish that the memetic algorithm is a new state-of-the-art algorithm for the LOP.

The paper is structured as follows. Section 2 gives an overview of the instance classes that we studied. Section 3 introduces the greedy algorithm and the local search techniques that are used by the metaheuristics. Details on the structural analysis of the benchmark instances is given in Section 4, while Section 5 describes the results of the search space analysis. Section 6 introduces the metaheuristics we applied and the following Section 7 gives a detailed account of the computational results. Finally, we conclude in Section 8.

2 LOP instance classes

So far, researches on the LOP made use of a number of different classes of benchmark instances, including instances stemming from real-world applications as well as randomly generated instances. However, typically not all the instances are tackled in all the available papers and, in addition, several of the randomly generated instances are not available publically.

The probably most widely used class of instances are those of LOLIB, a benchmark library for the LOP that comprises 49 real-world instances corresponding to input-output tables of economical flows in the EU. LOLIB is available at `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/`; for all LOLIB instances optimal solutions are known [11].

Our initial results with an ILS algorithm for the LOP, which were presented in [25], indicated that the LOLIB instances are actually too small to pose a real challenge to state-of-the-art metaheuristic approaches and even to exact algorithms. Therefore, we have generated through sampling the elements of the original matrix an additional set of large, random, real-life like instances that have a similar structure as those of LOLIB and that therefore should have the same or at least similar characteristics. We call this instance class XLOLIB for eXtended LOLIB. We generated for each instance of LOLIB two large instances, one of size $n = 150$ and the other of size $n = 250$, resulting in a total of 49 instances of size 150 and 49 instances of size 250. Initial tests showed that these instances are well beyond the capabilities of the exact algorithm by Mitchell and Borchers [22], one of the best performing exact algorithms. For example, on an instance of size 250 we aborted the program after one week computation time without identifying an optimal solution.

A real life instance consisting of a single input-output matrix of 81 sectors of the U.S. economy is available from the Stanford graph-base library [16], which is accessible at `http://www-cs-faculty.stanford.edu/~knuth/sgb.html`. From this instance, we generated several smaller instances by randomly selecting sectors. In fact, we created nine instances: three of 50 elements, three of 65, one of 79, one of 80 and one of 81 (that is the complete matrix). We refer to this class with SGB. Instances from the class SGB have been used in [17] and [9] to evaluate tabu search and iterated local search algorithms. There instances of size 40, 60, and 75 with 25 instances for each size were generated.

Because LOLIB instances are rather small, Mitchell and Borchers [22] generated large instances in their research on exact algorithms for the LOP. They generate a matrix by first drawing numbers between 0 and 99 for the elements in the upper triangular, and between 0 and 39 for the others, and then shuffling the matrix. This technique is used in order to obtain a linearity similar to the LOLIB instances; the linearity is the ratio of the optimal objective function over the sum of the matrix elements excluding those on the diagonal. Furthermore, zeros are added to increase the sparsity of the matrix. The range used to draw numbers is extremely limited when compared with the values that LOLIB instances elements can take. The idea underlying this choice is that there should be a large number of solutions with costs close to the optimal value, resulting in, according to Mitchell and Borchers, hard instances. Thirty of these instances with known optimal solutions are available at

`http://www.rpi.edu/~mitchj/generators/linord`, where also the generator can be found; we will refer to this instances as MBLB (Mitchell-Borchers LOP Benchmarks). Of these available instances, five are of size 100, ten of size 150, ten of size 200, and 5 of size 250. Even if the size of the MBLB instances is comparable to those of XLOLIB, preliminary tests showed that MBLB instances are significantly easier than XLOLIB instances. In fact, for all MBLB instances optimal solutions are known.

Finally, another class of randomly generated instances were proposed in [17]. There, 75 instances were generated using a uniform distribution in the range between 0 and 250000. Laguna, Martí, and Campos generated 25 instances each for the sizes 75, 150, and 200. We call this instance class LMC-LOP. These instances were made available by Rafael Martí.

3 Constructive and local search algorithms

The currently best known constructive algorithm for the LOP is due to Becker [3]. In a first step the index that maximizes the cost

$$q_i = \frac{\sum_{k=1}^{n} c_{ik}}{\sum_{k=1}^{n} c_{ki}} \quad i = 1 \ldots n$$

is chosen, and it is put in the first position of the permutation. Next, this index together with the corresponding column and row is removed and the new q_i values for the remaining indices are computed from the resulting sub-matrix. These steps are then repeated until the index list is empty, resulting in a computational cost of $O(n^3)$. A straightforward variation of this algorithm is to compute the q_i values only once at the start of the algorithm, sort these values in non-increasing order to yield a permutation of the indices. Using this variant, a solution can be computed in $O(n^2)$.

Both, the original algorithm and the variation, return good solutions compared to random ones. For example, the average deviation from the optimum solutions for LOLIB instances is 6.52 with the original algorithm, 9.46% with the variation, and 30.48% for random solutions; for MBLB the deviation obtained are 2.91% with the original algorithm, 2.52% with the static variation, and 40.34% for random solutions.

Better solutions than with Becker's construction heuristic are obtained with local search algorithms. We considered three possibilities. The first two are based on neighborhoods defined through the operations applicable to the current solution.

The first neighborhood, $\mathcal{N}_X$, is defined by the operation *interchange*; it is given as *interchange* : $\Pi \times \{1, \ldots, n\}^2 \rightarrow \Pi$, where Π is the set of all permutations and we have for $i \neq j$:

$$\textit{interchange}(\pi, i, j) \triangleq (\ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots)$$

This neighborhood has size $|\mathcal{N}_X| = n(n-1)/2$. Preliminary tests showed that $\mathcal{N}_X$ gives significantly worse results when compared to the following two local search methods.

A second neighborhood, $\mathcal{N}_I$, is defined by the *insert* operation: an element in position i is inserted in another position j. Formally, *insert* : $\Pi \times \{1,\ldots,n\}^2 \rightarrow \Pi$ is defined for $i \neq j$:

$$\textit{insert}(\pi,i,j) \triangleq \begin{cases} (\ldots,\pi_{i-1},\pi_{i+1},\ldots,\pi_j,\pi_i,\pi_{j+1},\ldots) & i<j; \\ (\ldots,\pi_{j-1},\pi_i,\pi_j,\ldots\pi_{i-1},\pi_{i+1},\ldots) & i>j; \end{cases}$$

The insert based neighborhood has size $|\mathcal{N}_I(\pi)| = (n-1)^2$.

The Δ-function

$$\Delta_I(\pi,i,j) \triangleq f(\textit{insert}(\pi,i,j)) - f(\pi)$$

associated with this operation is defined as:

$$\Delta_I(\pi,i,j) \triangleq \begin{cases} \sum_{k=i+1}^{j} c_{\pi_k\pi_i} - c_{\pi_i\pi_k} & i<j; \\ \sum_{k=j}^{i-1} c_{\pi_i\pi_k} - c_{\pi_k\pi_i} & i>j. \end{cases}$$

The cost for evaluating this function is $O(|i-j|)$ and, hence, in $O(n)$ if no care is taken. In a straightforward implementation of a local search based on this neighborhood one would, given an index i, tentatively try all possible moves *insert*(π,i,j) with j ranging from 0 to $n-1$. Since for each move the Δ-function evaluation is linear, the cost for exhaustively visiting the neighborhood is $O(n^3)$. However, the local search procedure can be sped up significantly, if the neighborhood is visited in a more systematic way. A particular case of an insert move is given if $i = j \pm 1$; we call this a *swap* move, and its Δ function is:

$$\Delta_S(\pi,i,j) \triangleq \begin{cases} c_{\pi_i\pi_j} - c_{\pi_j\pi_i} & i=j+1; \\ c_{\pi_j\pi_i} - c_{\pi_i\pi_j} & i=j-1. \end{cases}$$

Hence, the cost of the evaluation of Δ_S is constant. Furthermore, an *insert* move (with arguments i and j) is always equivalent to $|i-j|$ *swap* moves. For example, for $n=7$ and $i=2, j=5$ we have

$$\begin{array}{rll} \pi = & 1\,2\,3\,4\,5\,6 & \rightarrow \\ & 1\,3\,2\,4\,5\,6 \quad = \textit{insert}(\pi,2,3) & \rightarrow \\ & 1\,3\,4\,2\,5\,6 \quad = \textit{insert}(\pi,2,4) & \rightarrow \\ & 1\,3\,4\,5\,2\,6 \quad = \textit{insert}(\pi,2,5). & \end{array}$$

In the example, it can be noticed that all the permutations visited when transforming π by applying *insert*$(\pi,2,5)$ are in the *insert*-neighborhood of π and are always obtained by applying a swap move to the previous step. Hence, the idea is to use only *swap*-moves to visit the whole $\mathcal{N}_I$. For each index i we will apply all the possible moves *insert*(π,i,j) in two stages. First, with indices j that range from $i-1$ to 0 and then for indices j that vary from $i+1$ to $n-1$. In every stage a solution can be obtained from the previous visited one by applying a *swap* move. Hence, every solution in the neighborhood can be obtained in constant time and therefore the total computational cost for evaluating the insert neighborhood becomes $O(n^2)$. This technique was inspired by the method that Congram applies to Dynasearch [9]. In Fig. 1 we show the effect of

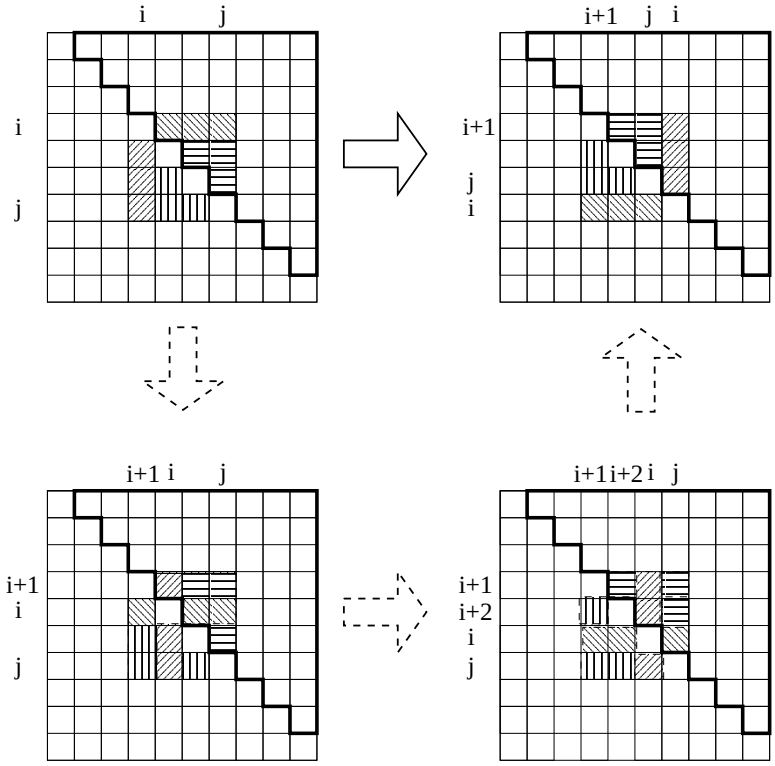

Figure 1: Given is a pictorial example of what the Δ function means in terms of matrix entries and how an insert move can be done with successive *swap* moves.

an *insert* move on the matrix and we also show how the same move can be done only through *swap* moves.

A further minor speed-up consists in pre-computing the cost for all the possible *swap* moves:

$$d_{ij} = c_{ij} - c_{ji} \quad \forall\, i, j = 0 \ldots n-1.$$

In addition to these standard neighborhoods, we also implemented the local search algorithm $\mathcal{CK}$ by Chanas and Kobylański [8] that uses two functions *sort* and *reverse*. When applied to a permutation, *sort* returns a new permutation in which the elements are rearranged according to a specific sorting criterion (see [8]), while *reverse* returns the reversed permutation. In the LOP case, if a permutation maximizes the objective function, the reversed permutation minimizes the objective function; hence, reversing a good solution leads to a bad solution. The idea of $\mathcal{CK}$ is to alternate sorting and reversing to improve the current solution; in fact, it has been shown that the application of *reverse* and then *sort* to a solution will lead to a solution with a value greater or equal the starting one. The functional description of the algorithm is:

$$(\mathit{sort}^* \circ \mathit{reverse})^* \circ \mathit{sort}^*$$

where $\circ$ denotes function composition, and the * operator is used to apply any given function iteratively until the objective function does not change. Formally, we consider a general function ϕ, and a generic permutation π:

$$\phi^*(\pi) \triangleq \begin{cases} \pi & f(\phi(\pi)) = f(\pi) \\ \phi^*(\phi(\pi)) & \text{otherwise} \end{cases}$$

The sort function is recursively defined as follows:

$$sort(\pi_1, \ldots, \pi_k) = \begin{cases} \pi_0 & k = 0, \\ insert_{\mathcal{CK}}(\pi_k, sort(\pi_0, \ldots, \pi_{k-1})) & k > 1 \end{cases} \quad (1)$$

$$insert_{\mathcal{CK}}(t, sort(\pi_0, \ldots, \pi_{k-1})) = (\pi_1, \ldots, \pi_{\bar{r}-1}, t, \pi_{\bar{r}+1}, \ldots, \pi_{k-1}) \quad (2)$$

where $\bar{r} \in 0, \ldots, k-1$ such that it maximizes the value:

$$\Delta_{\mathcal{CK}}(t, r, (\pi_0, \ldots, \pi_{k-1}) = \sum_{j=1}^{\bar{r}-1} c_{\pi_j t} + \sum_{j=\bar{r}}^{k-1} \pi_{t\pi_j}.$$

Unfortunately this definition does not help in understanding the neighborhood actually used by $\mathcal{CK}$. In fact, one can show that the $\mathcal{CK}$ algorithm actually implements a local search algorithms based on $\mathcal{N}_I$.

We implemented three local search variants, including two versions of the *insert* moves and the $\mathcal{CK}$. The two *insert* variants differ only in the pivoting rule applied. One version uses a pivoting rule that is somewhere between *first* and *best improvement*:

function $visit_{\mathcal{N}_I}(\pi)$
  for $i = 0..n-1$ **do**
    $\bar{r} \leftarrow \arg\max_{r.r \neq i} f(insert(\pi, i, r))$
    $\pi' = insert(\pi, i, \bar{r})$
    if $f(\pi') > f(\pi)$ **then**
      return(π'**)**
    end if
  end for
  return(π**)**

Obviously, the scan of the indexes for finding the best move for each index is made exploiting the evaluation of the delta function in constant time. We indicate with $\mathcal{LS}_f$ the local search on $\mathcal{N}_I$ based on the visit we just introduced; $\mathcal{LS}_i$ is a local search on $\mathcal{N}_I$ using a random first improvement strategy, where the neighborhood is scanned in a random order; the latter neighborhood examination scheme requires the Δ-function to be computed in linear time.

Table 1 gives a comparison of the three algorithms on the benchmark classes we considered. The results show that with respect to solution quality all three algorithms are comparable. However, they strongly differ in terms of computational speed. Clearly, $\mathcal{LS}_f$ is the fastest, followed by $\mathcal{CK}$; $\mathcal{LS}_i$ is several orders of magnitude slower than the other two. Based on these results, in the rest of the paper we will apply $\mathcal{LS}_f$ as local search.

		Avg.Dev. (%)	# optima	Avg.time (sec)
LOLIB	$\mathcal{LS}_i$	0.1842	42	0.1802
	$\mathcal{CK}$	0.2403	38	0.0205
	$\mathcal{LS}_f$	0.22	45	0.013
SGB	$\mathcal{LS}_i$	0.27	5	0.1389
	$\mathcal{CK}$	0.41	3	0.01013
	$\mathcal{LS}_f$	0.46	7	0.00516
MBLB	$\mathcal{LS}_i$	0.0195	10	9.81
	$\mathcal{CK}$	0.0209	12	0.22
	$\mathcal{LS}_f$	0.021	10	0.14
XLOLIB (250)	$\mathcal{CK}$	1.11	0	2.1256
	$\mathcal{LS}_f$	0.90	0	0.6741
LMC-LOP	$\mathcal{CK}$	0.65	0	1.0496
	$\mathcal{LS}_f$	0.60	0	0.2976

Table 1: Comparison between three local search algorithms on the benchmark classes. The results are averaged over all instances of each class and over 100 trials for each instance. Avg.Dev. gives the average percentage deviation from optimal or best known solutions, # optima gives the number of optimal or best known solutions found at least once in the 100 trials for each instance, and Avg.time (sec) gives the average computation time in seconds on a 1.4 GHz Athlon CPU to run the local search once on each benchmark instance of a class (for example, the timing given on LOLIB instances is the time to run a local search once for all the 49 instances of LOLIB).

4 Structural analysis of the instances

As a first step in our analysis of the LOP instance characteristics, we computed cross-statistical information on the distribution of the matrix entries for the available instances. In particular, we computed for all instances the sparsity, the variation coefficient and the skewness of the matrix entries. The sparsity measures the percentage of matrix elements that are equal to zero; the main interest in this measure is that according to Mitchell and Borchers, it has a strong influence on algorithm behavior [22]. The *variation coefficient* (VC) is defined as $\sigma/\bar{X}$, where σ is the standard deviation and $\bar{X}$ the mean of the matrix entries. VC gives an estimate of the variability of the matrix entries, independent of their size and their range. The *skewness* is the third moment of the mean normalized by the standard deviation; it gives an indication of the degree of asymmetry of the matrix entries. The statistical data is given in Table 2 for LOLIB and XLOLIB instances and in Table 3 for the random instance classes MBLB and LMC-LOP.

The cross statistical data for the SGB instances are that the median for the size 50 and 65 instances for the sparsity is 14.16 and 22.91, respectively, the VC is 4.59 and 5.23, respectively, and the skewness is 10.31 and 13.09, respectively. For the three instances of size 79, 80, and 81 these values are 26.26, 25.91, and 25.29 for the sparsity, 6.12, 6.13, and 10.62 for the VC and 16.62, 16.80, and 21.27 for the skewness.[1]

[1]Note that the full SGB instance of size 81 has the particularity of having a negative row, hence resulting

Table 2: Structural information on the real-world and real-world like instance classes. "Sp." indicates the sparsity, "VC" the variation coefficient, and "Sk." the skewness. Given are the minimum, the 0.25 and 0.75 quantiles, the median, the maximum and the mean of these measures across all instances of a benchmark class.

LOLIB

Size		Min	1st qu.	Median	3rd qu.	Max	Mean
all	Sp.	11.00	26.91	35.28	46.13	80.63	37.34
	VC	4.10	4.45	4.87	5.78	16.25	5.49
	Sk.	9.15	11.40	12.93	15.83	39.18	15.50

XLOLIB

150	Sp.	10.57	26.80	34.71	45.74	80.351	37.25
	VC	4.04	4.46	4.84	5.54	16.05	5.42
	Sk.	8.94	11.09	12.49	15.78	42.62	15.04
250	Sp.	10.79	26.98	35.04	45.76	80.48	37.25
	VC	4.07	4.39	5.00	5.77	15.81	5.48
	Sk.	9.05	11.33	12.61	16.51	43.63	15.49

These statistical data show that there are significant differences between the real-life instances (LOLIB and SGB) and real-life like random problems (XLOLIB) on the one side and the randomly generated instances from LMC-LOP and MBLB on the other side. For the real-life and real-life like instances all statistics (sparsity, VC, and skewness) are typically much higher than for the randomly generated instances. This suggest that the former class of instances are much less regular and the variation among the matrix entries is much stronger than for the random instances. Additionally, the variation of the statistics among the real-world (like) instances is much larger indicating a certain diversity of structural features in these instances. Obviously, SGB instances are an exception in that respect, because they are all generated from the same matrix. Differently, the variance of the statistical measures is low for LMC-LOP and MBLB instances, indicating a more regular structure of these.

The data presented here give evidence that we might observe significant differences in the behavior of algorithms when applied to random instances or real-life (like) instances. Additionally, these data give an indication that conclusions obtained for the random instances do not necessarily apply to real-life instances, because random instances show such different statistical data from real-life instances. Hence, the XLOLIB instances appear to be much better suited for testing algorithms on realistic, large LOP instances than the random instances.

in a somewhat different structure of this instance with respect to VC and skewness than other instances of similar size.

Table 3: Structural information on randomly generated instance classes. "Sp." indicates the sparsity, "VC" the variation coefficient, and "Sk." the skewness. Given are the minimum, the 0.25 and 0.75 quantiles, the median, the maximum and the mean of these measures across all instances of a benchmark class.

LMC-LOP

Size		Min	1st qu.	Median	3rd qu.	Max	Mean
75	Sp.	0.5	0.5	0.50	0.51	0.51	0.51
	VC	0.70	0.71	0.71	0.71	0.72	0.71
	Sk.	0.389	0.40	0.40	0.40	0.42	0.40
150	Sp.	1.33	1.33	1.33	1.33	1.37	1.34
	VC	0.70	0.71	0.72	0.72	0.73	0.72
	Sk.	0.36	0.40	0.41	0.43	0.46	0.41
200	Sp.	0.67	0.67	0.67	0.68	0.68	0.67
	VC	0.71	0.71	0.71	0.71	0.72	0.71
	Sk.	0.38	0.34	0.40	0.41	0.43	0.40

MBLB

100	Sp.	22.01	22.12	22.33	22.44	23.36	22.45
	VC	1.00	1.01	1.01	1.01	1.02	1.01
	Sk.	0.98	0.98	1.00	1.00	1.00	0.99
150	Sp.	2.29	2.38	7.15	12.02	12.48	7.23
	VC	0.77	0.78	0.83	0.88	0.89	0.83
	Sk.	0.82	0.84	0.86	0.88	0.88	0.86
200	Sp.	2.08	2.25	7.02	11.87	12.10	7.06
	VC	0.77	0.78	0.83	0.88	0.88	0.83
	Sk.	0.82	0.84	0.86	0.88	0.89	0.86
250	Sp.	2.04	2.11	2.12	2.15	2.23	2.13
	VC	0.77	0.77	0.78	0.78	0.78	0.77
	Sk.	0.82	0.83	0.84	0.84	0.84	0.84

5 Landscape Analysis

The central idea of the landscape analysis in combinatorial optimization is to represent the space searched by an algorithm as a landscape formed by all feasible solutions, which in the LOP case are permutations, and a *fitness* value assigned to each solution, which in our case is the objective function value $f(\pi)$ of a permutation π and to impose a distance metric on the search space [20]. The usefulness of this paradigm is typically based on (i) the insights with respect to search space characteristics and the relationship to the behavior of local search algorithms or metaheuristics [5, 31], (ii) the possibility to predict problem or problem instance difficulty [2, 28], or (iii) indications on useful parameterizations of local search algorithms [1].

Formally, the search landscape of the LOP is described by a triple $\langle \Pi(n), f, d \rangle$, where Π is the set of all permutations of the integers $\{1, \ldots, n\}$, f is the cost function

and d is a distance measure, which induces a structure on the landscape. It is natural to define the distance between two permutation π and π' in dependence of the "basic operation" used by a local search algorithm; typically, the distance then is given by the minimum number of applications of this basic operation needed to transform π into π'. Since the best performing local search algorithms are all based on the $\mathcal{N}_I$, we consider an *insert* move as our basic operation. Unfortunately, as far as we know, there is no efficient, that is polynomial, way of computing the minimum number of *insert* moves needed to transform one permutation into another one. Therefore, we use a surrogate distance that is based on the *precedence metric* [24]: for all pairs of elements j and i we count how often j precedes i in both permutations and then subtract this quantity from $n(n-1)/2$, which corresponds to the maximum possible distance.

5.1 Landscape correlation analysis

The first feature of the search landscape we studied is its ruggedness: a search landscape is said to be rugged if there is a low correlation between neighboring points. To measure this correlation, Weinberger suggested to perform a *random walk* in the search landscape of length m, to interpret the resulting set of m points $\{f(x_t)\}, t = 1, \ldots, m$ as a time series and to measure the autocorrelation $r(s)$ of points in this time series that are separated by s steps [33] as

$$r(s) = \frac{\sum_{t=1}^{m-s}(f(x_t) - \bar{f})(f(x_{t+s}) - \bar{f})}{\sigma^2(f)(m-s)},$$

where $\sigma^2(f)$ is the variance of the time series, and $\bar{f}$ its mean. Often, the resulting time series can be modeled as an autoregressive process of order one, and then the whole correlation structure can be summarized by $r(1)$ or, equivalently, by the *search landscape correlation length* that is computed as $\ell = -\frac{1}{ln(|r(1)|)}$ $(r(1) \neq 0)$ [26, 27, 33]; the lower is the value of ℓ, the more rugged is the landscape. Interestingly, in landscape analysis literature general intuitions and some results suggest that there is a negative correlation between ℓ and the hardness of the problem [2].

We computed ℓ on all benchmark instances with a random walk of one million steps; Table 4 summarizes data collected on all instances. ℓ is given normalized by the instance size, that corresponds also to the diameter of search landscape based on the $\mathcal{N}_I$ neighborhood. As we see, each class has a relatively small variance (SGB has one instance that represents an outlier in these data). This means, that the landscape correlation length can characterize specific instance classes. From these instance classes, the MBLB instances have the by far largest ℓ, which would suggest that these instances are also the easiest to solve; in fact, when abstracting from instance size, our experimental results with metaheuristic suggests that this is true. The next smaller ℓ is found for the real-life instances from LOLIB and SGB, while the smallest values, on average, are observed for LMC-LOP and the XLOLIB instances.

Regarding instance class definitions, note that the values of ℓ/n alone are not sufficient. For example, XLOLIB instances showed roughly the same normalized values for ℓ/n as LMC-LOP instances, which would suggest similar behavior. However, both types of instances have widely different characteristics as shown by the data on the distribution of the matrix entries.

Table 4: Given are standard statistical data (minimum, 0.25 and 0.75 quantiles, median, average, and maximum) for the normalized values ℓ/n of the search landscape correlation length measured across all the instances of the available benchmark classes.

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.7536	0.7907	0.8004	0.8207	0.8403	0.8021
SGB	0.4821	0.8055	0.8163	0.8248	0.8347	0.7810
XLOLIB(100)	0.7094	0.7278	0.7311	0.7416	0.7671	0.7341
XLOLIB(250)	0.7165	0.7327	0.7364	0.7407	0.7524	0.7372
MBLB	0.9339	0.9595	0.9639	0.9707	0.9775	0.9620
LMC-LOP	0.6924	0.7211	0.7312	0.7450	0.7746	0.7332

From the methodological point of view we were interested on how long a *random walk* should be to obtain a stable estimate of ℓ. Therefore, we measured on all MBLB and XLOLIB instances of size 250 ten times ℓ for different lengths of the random walks. Figure 2 shows the ℓ we found in these experiments for all the instances. As we see, the longer is the random walk the more precise is the resulting measure of ℓ. These plots also indicate that apparently for 1.000.000 steps in the random walk the ℓ estimate has stabilized. On the other side, these results also suggest, that the random walks for measuring ℓ should be a large multiple (e.g. 400 in this case) larger than the instance size to result in stable estimates.

5.2 Fitness-distance analysis

In a next step we analyzed the distribution and the relative location of local optima to the global optima of the LOP.

For LOLIB we run 13,000 local searches starting from random solutions, while for the other instance classes 1,000 local searches were done. On the instance classes LMC-LOP and XLOLIB the local searches generated 1,000 distinct local optima for each instance and in no case the best known solutions were found. For LOLIB instead the number of distinct local optima was considerably varying, between 24 and 13000 with a median around 9400; for MBLB the number ranged from around 73 and 1000, with a median of 965; finally for the smaller SGB instances around 600 distinct local optima were found, while in the largest ones 1,000 distinct local optima resulted. Summary data are given also in Table 5.

For all LOLIB, SGB and MBLB instances we know the global optima. In fact, on several instances we could identify also global optima among the local optima generated. Among the total number of distinct local optima, the percentage of global optima ranges from 0.47% to 85.12% for LOLIB, while for the other instance classes the corresponding percentages are much smaller. Summary data on these values are given in Table 6. In fact, these results also suggest that especially the LOLIB instances can effectively be solved by a random restart algorithm that is run long enough.

Finally, we analyzed the relationship between the quality of local optima and their distance to the closest global optimum by measuring the fitness distance correlation coefficient and measuring fitness distance plots [13]. Given a sample of m candidate

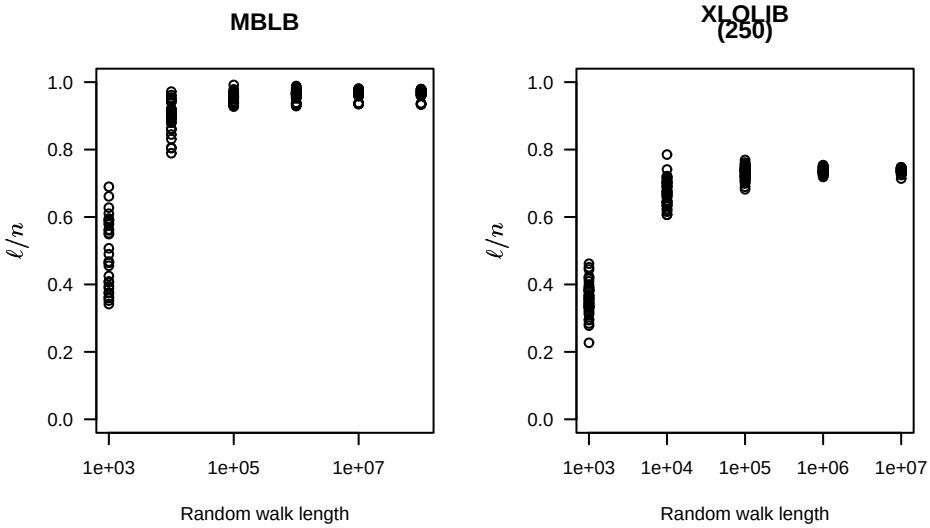

Figure 2: Dependence of ℓ/n (given in the y-axis) on the *random walk* length (x-axis) for MBLB (left) and XLOLIB (right) instances. Every dot is the average normalized landscape correlation length measured across 10 *random walks* of the corresponding length for one instance of the class.

Table 5: Summary information on the percentage of distinct local optima found (percentage of the total number of local optima generated).

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.12	40.35	63.65	80.49	99.2	67.93
SGB	25.60	64.50	94.60	99.5	100.00	81.09
MBLB	7.30	76.75	96.50	99.48	100.00	82.17

solutions $\{\pi_1, \ldots, \pi_m\}$ with an associated set of pairs $\{(f_1, d_1), \ldots, (f_m, d_m)\}$ of fitness (solution quality) values f_i and distances to the closest global optimum d_i, the (sample) fitness distance correlation coefficient ρ can be computed as

$$\rho = \frac{\widehat{Cov}(f, d)}{\widehat{\sigma}(f) \cdot \widehat{\sigma}(d)} \tag{3}$$

where

$$\widehat{Cov}(f, d) = \frac{1}{m-1} \sum_{i=1}^{m} (f_i - \bar{f})(d_i - \bar{d}), \tag{4}$$

Table 6: Summary information on the number of distinct global optima found, given as the percentage of the total number of distinct local optima.

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	0.47	2.34	14.10	16.54	85.12	8.33
SGB	0.00	0.10	0.20	1.17	1.86	0.68
MBLB	0.00	0.00	0.22	0.57	4.27	0.46

Table 7: Some information on the local optima generated, the mean distance from the best known solution is given as percentage over the max distance. The number of distinct local optima is given as percentage of the number of local optima generated; the LOLIB value is over 13000 trials explaining in part the very low value.

Class	Avg.Dist (%) from best known	Avg. (%) distinct local optima
LOLIB	5.84	58.55*
SGB	10.51	81.18
XLOLIB	26.27	100
MBLB	0.43	82.02
LMC-LOP	23.22	100

$$\widehat{\sigma}(f) = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(f_i - \bar{f})^2}, \qquad \widehat{\sigma}_F = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(d_i - \bar{d})^2}, \qquad (5)$$

and $\bar{f},\ \bar{d}$ are the averages over the sets $F = \{f_1, \ldots, f_m\}$ and $D = \{d_1, \ldots, d_m\}$, respectively. $\widehat{Cov}(f, d)$ is the sample covariance between the f and d values, while $\widehat{\sigma}_F$ and $\widehat{\sigma}_D$ are the sample standard deviations of F and D, respectively. As usual, we have $-1 \leq \rho \leq 1$. In our case, we used as the fitness the deviation from the global optimum. Hence, a high, positive value of ρ indicates that the higher the solution quality, the closer we get to global optima, on average and, hence, the solution quality gives good guidance when searching for global optima. For the instances of the classes LMC-LOP and XLOLIB we do not have proven optimal solutions available, since exact algorithms were not able to solve these. In this case, we used the best known solutions instead of global optima. The best known solutions were the best ones found by the metaheuristics we tested in Section 7. In the case of the LMC-LOP instances of dimension 75, these best known solutions are conjectured to be the optimal ones, since the same best solutions were found in many trials of the metaheuristics we tested.

In addition, we used a second measure of the FDC that is variation on the original measure. For this new measure the FDC is computed only for the local optima with an objective function value that is better than the median objective function for all the local optima that were generated. The idea behind this measure is that all the metaheuristic should be able to easily reach a solution with such an objective function value. In fact, by a simple random restart, within few iterations the probability of finding a solution better than a median local optimum approaches one. Furthermore, in an analysis of

Table 8: Summary information for ρ, the fitness distance correlation coefficient, computed on the complete set of local optima.

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	-0.1056	0.4901	0.6763	0.7833	1.0000	0.6189
SGB	0.2750	0.6297	0.6674	0.7466	0.9177	0.6409
XLOLIB	0.2412	0.3925	0.4520	0.4907	0.6513	0.4484
MBLB	0.6144	0.7224	0.7948	0.8423	0.9395	0.7867
LMC-LOP	0.2876	0.4857	0.5760	0.6493	0.8193	0.5662

Table 9: Summary information for ρ', the easy level fitness distance correlation coefficient, computed on the local optima better than the *easy level* (see text for details).

	Min	1st qu.	Median	3rd qu.	Max	Mean
LOLIB	-0.7214	0.3117	0.5844	0.7493	0.9698	0.4573
SGB	0.1910	0.4850	0.5977	0.7247	0.8609	0.5774
XLOLIB	0.1593	0.2736	0.3245	0.3696	0.5423	0.3219
MBLB	0.2480	0.5421	0.6349	0.7031	0.8074	0.6123
LMC-LOP	0.06305	0.34620	0.45340	0.57380	0.79050	0.45900

the FDC relationship one should focus on the solutions which are the more likely ones to be encountered in the search trajectory of the metaheuristics. We will refer to the threshold on the solution quality as the *easy level* and to the FDC based on the solutions passing this bound as the *easy level FDC* (ρ').

Table 8 and 9 summarize the information about FDC and *easy level FDC*, respectively. In general, it appears that the easy level FDC coefficients are lower than the standard FDC coefficients. This probably is the case because poor quality local optima that are far from the global optimum can have a considerable influence on the resulting correlation and these poor local optima are eliminated when imposing the "easy level" bound. For some instance the values of ρ and ρ' are strongly positive, suggesting that these instances should be relatively easy for restart type algorithms [20].

In Figure 3 we show some example FDC plots for instances from all benchmark classes except XLOLIB. Since the range of the x-axis is from zero to the maximum distance, the plots give visual information on the typical distance of local optima to the nearest global optima (see also Table 7). As we see for all the problems the local optima are close to the global optima (or best known solutions), typically the average distance is less than a third of the maximal distance. On the extreme side are the MBLB instances, for which all the local optima are extremely close to a global one, the maximal distance over all instances we observed was 191, which corresponds to 3.86% of the maximum distance. To give a more detailed picture of the fitness-distance relationship for the MBLB instances, we plot in Figure 4 the same data as in Figure 3 but using a logarithmic scale on the x-axis.

In Section 7, we will give an analysis of how the fitness distance measure correlates with the hardness of LOP instances for particular metaheuristics.

According to the FDC analysis and the FDC plots, the MBLB instances would be

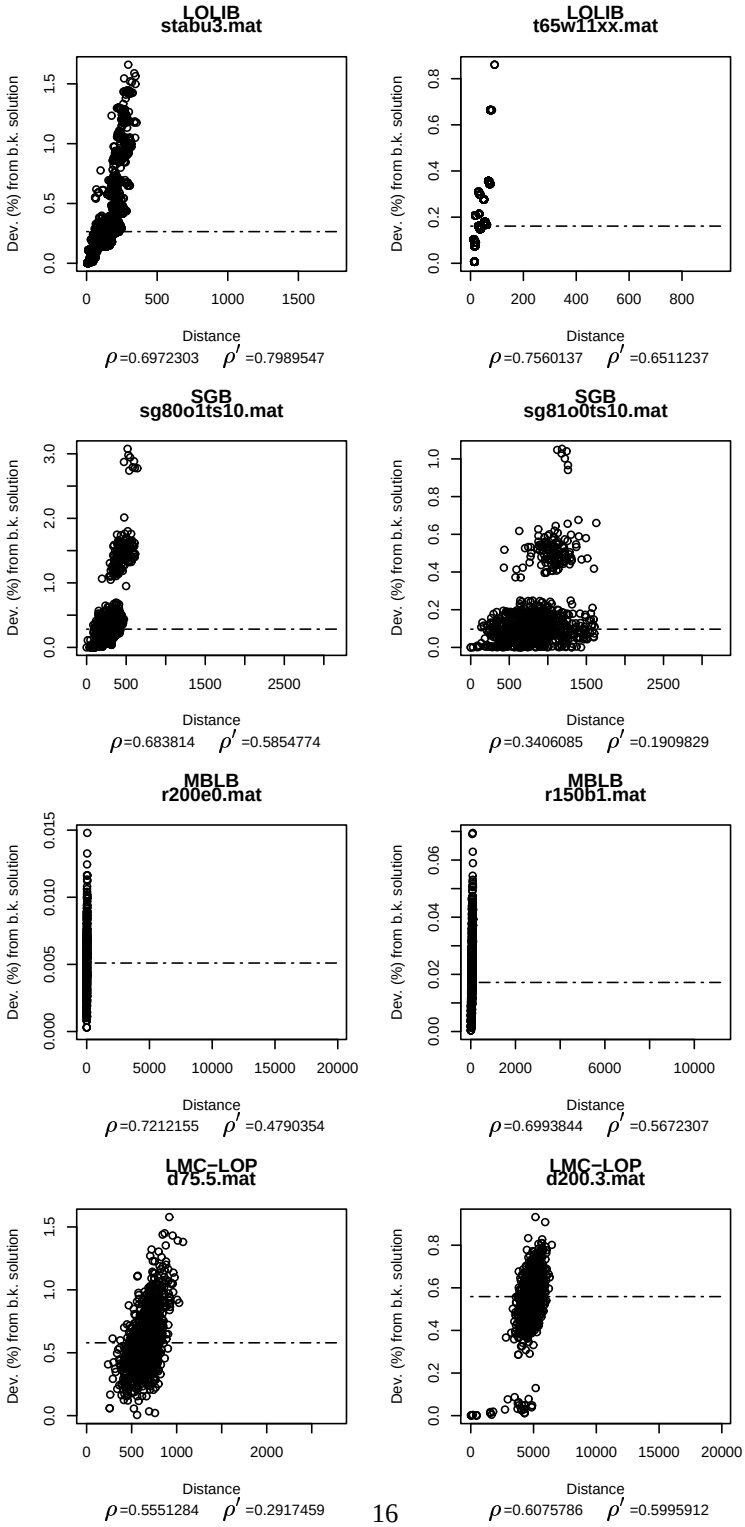

Figure 3: Examples of FDC plots. On the x-axis is given the distance to the nearest global optimum (or best known solution if optima are not proven) and on the y axis the percentage deviation from optimum or best known solution. The dashed line indicates the median deviation from the best known or globally optimal solution over the randomly generated local optima.

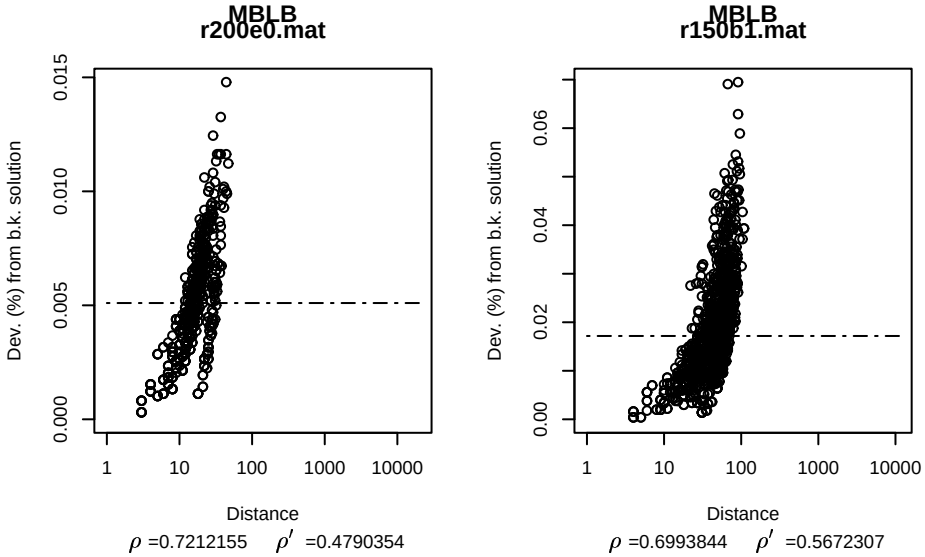

Figure 4: FDC plots for two MBLB instances; here the x-axis uses a log scale.

predicted to be the easiest ones, when abstracting from pure instance size. This is the case because of the very high FDC coefficients and the concentration of the local optima in a very tiny part of the whole search space. A further confirmation of this impression is given by the analysis of the landscape ruggedness through the correlation length of random walks. In fact, later experimental results suggest that MBLB instances are actually easily solved, while XLOLIB and LMC-LOP instances of a similar size to the MBLB instances are by far harder to solve for metaheuristics but also for exact algorithms. An additional, interesting observation is that the landscape analysis would suggest that the characteristics of the XLOLIB instances are slightly different from the original LOLIB instances, differently from the structural analysis of these instances. In fact, the later experimental evaluation showed that XLOLIB instances are much harder to solve than LOLIB instances (see also experimental evaluation in Section 7) and this observation gives an indication that this result may not only be due to their larger size.

6 Metaheuristics

The results of the search space analysis of the LOP suggest that methods that are able to exploit both the good performances of the local search, and the often highly positive fitness distance correlation are promising for this problem. Earlier research results suggest that two metaheuristics that have these characteristics are Iterated Local Search (ILS), and Memetic Algorithms (MAs) [5, 20, 19].

Algorithm 1 Algorithmic outline of an ILS algorithm.

$\pi \leftarrow$ *GenerateInitialSolution*();
$\pi \leftarrow$ *LocalSearch*(π);
repeat
  $\pi' \leftarrow$ *Perturbation*(π);
  $\pi' \leftarrow$ *LocalSearch*(π');
  $\pi \leftarrow$ *AcceptanceCriterion*($\pi, \pi', history$);
until termination condition met;

6.1 Iterated Local Search

Iterated local search (ILS) is a conceptually very simple but at the same time very powerful, metaheuristic, as shown by a number of available applications results [18]. ILS iterates in a particular way over the local search process by applying three main steps: (i) perturb a locally optimal solution, (ii) locally optimize it with the local search chosen and (iii) choose, based on some acceptance criterion, the solution that undergoes the next perturbation phase. Algorithm 1 describes the general algorithmic outline for ILS. Next, we indicate the possibilities we considered for the final ILS algorithm.

- *GenerateInitialSolution*: The initial solution is taken to be a random permutation.
- *LocalSearch*: The local search procedure is the core of the algorithm and the overall ILS performance depends strongly on it. For the ILS algorithm, we use the $\mathcal{LS}_f$ local search, which was the best performing according to Section 3.
- *Perturbation*: As perturbation operator we used *interchange* moves, because it is a move that cannot be undone by insert moves in one step. The number of *interchange* moves to be applied in a perturbation is a parameter of the algorithm.
- *AcceptanceCriterion*: It determines to which solution the next perturbation is applied. We tried different approaches, the final choice of which one to be used was made using an automatic tuning procedure.

  Accept better: A new local optimum is accepted only if the objective function is larger than the current best solution, that is, is $f(\pi') > f(\pi)$;

  Accept small worsening: A new local optimum is accepted if the objective function $f(\pi')$ is larger than $(1 - \epsilon) * f(\pi)$, where ϵ is a parameter to be tuned;

  Simulated annealing like: We apply a probabilistic acceptance/rejection test based on the standard Metropolis acceptance criterion in simulated annealing [15]. In this case, the parameters to be tuned are the initial temperature, the temperature cooling ratio, and the number of step between consecutive temperature reductions.

6.2 Memetic Algorithm

Memetic algorithms are evolutionary algorithms that are intimately coupled with local search algorithms, resulting in a population-based algorithm that effectively searches

Algorithm 2 Algorithmic outline of an memetic algorithm.

```
Population ← {};
for i=1...m do {m is the number of individuals}
  π ← LocalSearch(GenerateRandomSolution());
  Population ← Population ∪ {π};
end for
repeat
  Offsprings ← {};
  for i ← 1...#crossovers do
    draw π_a, π_b from Population
    Offsprings ← Offsprings ∪ {LocalSearch(Crossover(π_a, π_b))};
  end for
  for i ← 1...#mutations do
    draw π_a from Population
    Offsprings ← Offsprings ∪ {LocalSearch(Mutate(π_a))};
  end for
  Population ← SelectBest(Population ∪ Offsprings, m);
  if same average solution quality for a long time then {diversification}
    Population ← SelectBest(Population, 1);
    for i=1...m-1 do {m is the number of individuals}
      π ← LocalSearch(GenerateRandomSolution());
      Population ← Population ∪ {π};
    end for
  end if
until termination condition met;
```

in the space of local optima [23].

Algorithm 2 shows the algorithmic scheme of MAs that we used in our implementation. In the first step a *population* of *individuals* is obtained by first generating m distinct random permutations and applying to each $\mathcal{LS}_f$. Then, in each iteration (*generation*) a number of new individuals are created by applying *crossover* and *mutation* operators (in the literature these new individuals are called *offsprings*). The individuals to which *crossover* and *mutation* are applied are chosen randomly according to a uniform distribution as in several other, high performing MAs [19]. The crossover operator takes two individuals of the current population and combines them into a new individual, while the mutation operator introduces a perturbation into an individual. To each of the offspring $\mathcal{LS}_f$ is applied. Finally, the best m individuals from the original population and the newly generated ones are selected for the new population; care is taken to eliminate duplicates.

In addition to this rather standard scheme for MAs, we use a diversification mechanism that is triggered if the average objective function of the population has not changed for a number of steps. In this case, we generate a new, random initial population, keeping only the overall best individual.

It is well known that the performance of an MA may depend strongly on the cross-

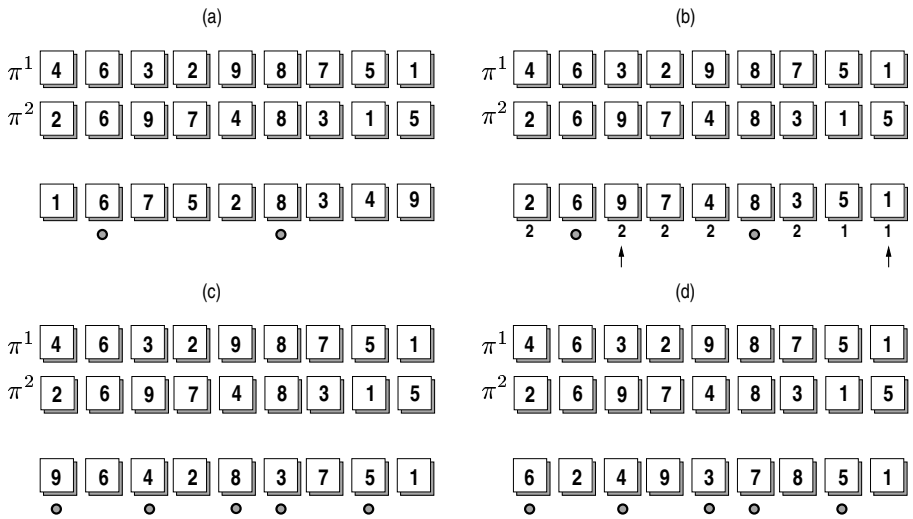

Figure 5: (a) DPX operator: the positions marked with a circle are common in the parents; (b) CX operator: the positions marked with a circle are common in the parents (c) OB operator: the positions marked with a circle are the reordered elements ($k = 5$); (d) Rank operator

over operator. Therefore, we tested four different ones.

DPX (Fig. 5a): The offspring inherits the elements that have the same position in both parents; these are put in the same position as in the parents. The others element are assigned randomly between those position that are not chosen yet. This results in an offspring that, on average, has a same distance from both parents.

CX (Fig. 5b): The idea of CX is to keep as much information as possible from the parents. For the elements in common between the parents it works like the previous operator. For the others, the CX operator chooses randomly an empty position (i) and a parent (π^1), determining an element a ($a = \pi^1_i$) that in turn is assigned to the offspring in position i. In the second parent, π^2, a different element $b = \pi^2_i$ occupies this same position i. The element b is then copied to the offspring in the position occupied by b in π^1. This process iterates until all the position are filled (see [21]).

OB (Fig. 5c, order based): In the first phase the solution of the first parent is copied to the offspring. In the second phase it selects k positions, $0 < k < n$, and orders the elements in these k positions according to their order in the second parent (see [32]);

Rank (Fig. 5d): The offspring permutation is obtained sorting the elements by their average ranking over the two parents, ties are broken randomly according to a uniform distribution.

6.3 Parameter tuning

The tuning of the ILS and the MA algorithm was done in a systematic, statistically well-funded approach. We have developed a number of different candidate configurations for the two algorithms, 78 in the ILS case and 144 in the MA case, and a final configuration was selected using an automatic tuning procedure based on F-races [4]. The F-race returns the configuration of a metaheuristic, that performs best with respect to some evaluation metric on a number of instances that are used for parameter tuning. In our case, parameter tuning was done using XLOLIB instances of size 100. Notice that the instances used for tuning are different from the solutions in the benchmark sets on which the computational results are presented in the following. Hence, we have a clear separation of the instances into training instances, used for parameter tuning and test instances, on which the final results are presented.

Certainly, it may be argued that tuning the algorithms on one specific instance class and testing them in possibly different ones may give a bias in the results. However, this procedure gives also an impression of the robustness of an algorithm, since we can examine how the performance on one instance class generalizes to a wider set of instances. Additionally, we did further experiments testing different configurations when deemed necessary (see, for example, Section 7.3), so that a more complete picture of the overall performance can be obtained.

In the ILS case, the tuning concerned mainly the acceptance criterion to be used and the strength of the perturbation. The final configuration returned uses the acceptance criterion that accepts slightly worsening solutions with $\epsilon = 0.0001$ and the perturbation consists of 7 *interchange* moves.

In the MA case, we performed some exploratory experiments before applying the actual tuning procedure. In this preliminary experiments we found that the CX and OB crossovers gave best results. Therefore, we considered only these two in the automatic tuning phase. Mutations are obtained by applying the $interchange$ operation on an individual. The number the operation is applied is a parameter; we fixed this parameter to 7, so that the mutation corresponds to the perturbation in the ILS case. In the actual tuning phase with the F-races we considered 144 different configurations (more than for ILS, but ILS has few parameters). We compared the one that resulted to be the best to the two extreme cases that are an MA without crossover and one without mutations. This final comparison let to the choice of the configuration with no mutation, supposedly, because the partial restarts were enough to introduce diversification into the search, making the mutation unnecessary. The final configuration had a population size of 25 individuals, every generation 11 new offsprings are formed using the OB crossover operator, and the diversification was applied after the average fitness of the population has not changed for 30 generations.

7 Experimental results

In this section we study the performance of ILS and MA and compare them to results from the literature. The algorithms were run on dual processor Athlon 2400+ machine with a clock speed of 2GHz and with 1GB of RAM; since the algorithms were imple-

mented as single processes (no threads) only one processor was used at time. Each algorithm has been run 100 times on each instance from LOLIB, MBLB, and SGB and 30 times on the XLOLIB and the LMC-LOP instances. For the experiments a CPU time limit of 120 seconds was fixed, only runs that reached the known optimum solution value were aborted prematurely.[2]

The performance analysis is divided into two parts, in the first one we consider the instances for which we know the optimal or the conjectured optimal objective function value, while in the second the other instances are considered. Next, different aspects of the algorithms are analyzed in more detail and the results are compared to literature.

7.1 Instances with known optimal solutions

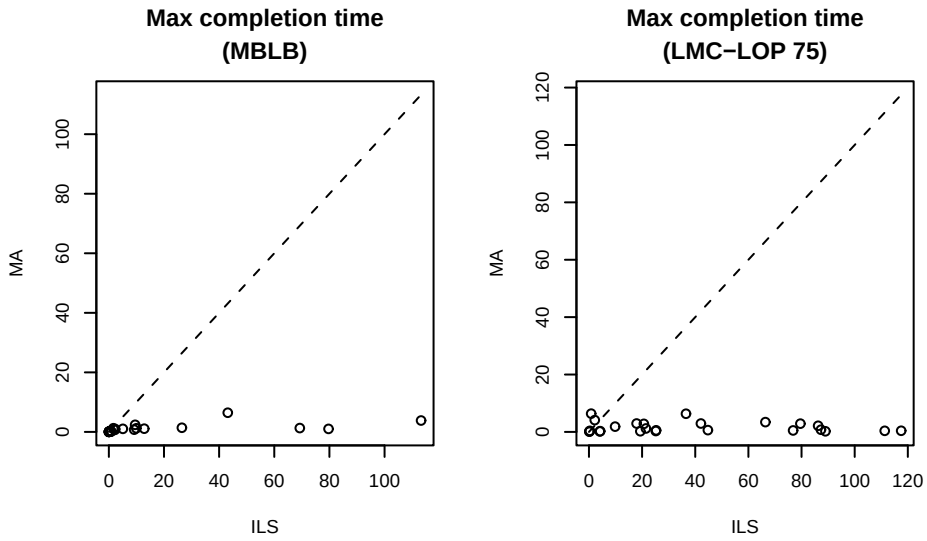

Figure 6: Pairwise comparison of ILS and MA with respect to the maximum time taken to solve MBLB (left) and LMC-LOP (right) instances. Each point corresponds to one instance; the x-axis indicates the maximum computation time of ILS and the y-axis gives the maximum computation time for the MA.

For all LOLIB, SGB, MBLB instances we were able to determine the optima, using an exact algorithm that is presented in [22]. The same algorithm is not able to find solutions for the instances of the other classes in reasonable computation time (this was tested running randomly chosen instances for 18 hours). For the instances of size 75 in LMC-LOP, we observed that the MA was finding for each instance a same objective function value; the very same value was the best found by ILS, which, however, did

[2]To give some impression of the time limits, let us state that the time taken to apply 1000 time $\mathcal{LS}_f$ starting from random solutions on the `be75eec_250.mat` instance from XLOLIB of size 250 took 5.86 seconds.

not reach such a solution in every single trial. Therefore, we strongly suppose that this value is optimal and we used it as (conjectured) global optimum for our analysis. In fact, the generated solutions were found to be unique. Solutions that we conjecture to be global optima will also be called pseudo-optima in the following.

Experiments with MA and ILS showed that the LOLIB and SGB instances are extremely easy and cannot be considered as challenging benchmarks for state-of-the-art metaheuristics. In fact, the longest of the 100 trials by ILS and MA for any of the instances of LOLIB took less than 0.1 seconds and less than 0.2 seconds for the SGB instances.

For MBLB the situation is different. While the MA obtained very good results and the maximum time to solve an instance was 6.5 seconds, the results for ILS were much worse, as shown in Figure 6 on the left. In fact, for 2 instances the ILS algorithm failed to find an optimum in all the runs (once in one instance, three times for the other one) and when it succeeded, it took much longer than the MA.

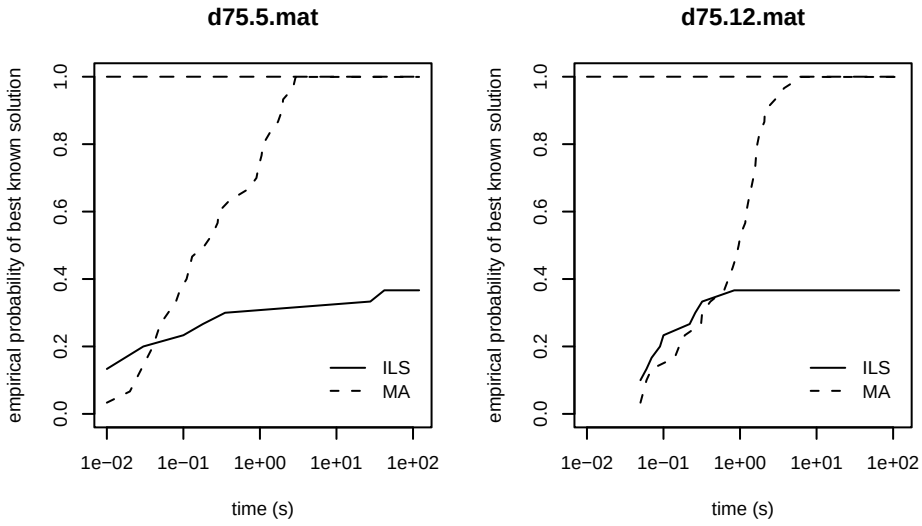

Figure 7: Run time distributions for the hardest LMC-LOP instances of size 75. Given is on the x-axis the computation time and on the y-axis the empirical cumulative probability of finding a pseudo-optimal solution.

Somewhat stronger tradeoffs between MA and ILS performance became noticeable on the LMC-LOP instances of size 75 (Figure 6, right). Here, the MA obtained much better results than ILS: First, as said above, the MA was able to find the best known solutions in all the 30 runs, while ILS could reach the same level of performance only for 12 instances. For seven instances, it found the best known permutation even in less than 50% of the runs.

To give a more detailed impression of the behavior of the MA and the ILS algorithm, for selected instanced we examined the run-time distributions that give the

empirical probability of finding a global optimum (or a bound on the solution quality) in dependence of the computation time [12]. Figure 7 gives the RTDs for the two hardest LMC-LOP instances of size 75 as judged by the computational results of ILS. In both cases, ILS finds the best known solution in only 11 out of 30 trials. The plots show that (i) the probability of finding such a solution for the MA increases continously and quickly reaches one, suggesting that MA is preferable to the ILS for computation times in the range of a few seconds and (ii) that the ILS algorithms suffers from a type of stagnation behavior that strongly compromises its performance.

In fact, the observation of search stagnation suggests that ILS performance can be strongly improved by including additional means of search diversification [30]. Therefore, we introduced a simple diversification step for the ILS that restarts the algorithm from a new random solution, if no improved solution is found for a number n_{ni} of iterations. The parameter n_{ni} was set (without tuning) to 750, which is the same number of solutions visited by the MA before the diversification step is applied. The new algorithm (ILS_R) strongly improved over the "standard" ILS, although it did not fully reach the level of performance of the MA. Certainly, ILS_R could be improved by additional tuning or by using more sophisticated ways of search diversification in ILS.

Finally, we compare the computational results of MA to an exact algorithm by Mitchell and Borchers (SimpMB) [22], which is based on the Simplex algorithm and that uses a branch and bound procedure to find an integer solution. Figure 8 gives a visual comparison of the results on the LOLIB and MBLB instances. MA is clearly much faster than SimpMB, often by several orders of magnitude. The much superior performance of MA over SimpMB is further confirmed by the fact that SimpMB was not able to solve any instances of XLOLIB or LMC-LOP within many hours of computation time.

7.2 Instances with unknown global optima

For the XLOLIB and the LMC-LOP instances with matrices of dimension larger than 75 no optimal or pseudo-optimal solutions are known. Therefore, we restricted the comparison of MA and ILS to a statistical analysis of the quality of the solutions returned by the two algorithms after the maximum computation time. The overall result was that for the large instances of LMC-LOP, the MA obtains average results that are significantly better than those of ILS, as confirmed by a Wilcoxon test with $\alpha = 0.01$. Similarily, the MA gives better performance on the XLOLIB instances. This is true for the instances of dimension 150, as indicated by a Wilcoxon test with $\alpha = 0.01$; however, for the large instances of dimensionality 250, no significant differences between the MA and the ILS could be found. The computational results are visualized in Figure 9.

7.3 Tuning effect for ILS

The results presented in Section 7.1 for ILS on the MBLB instances are actually much worse than those presented in an earlier article with a different ILS algorithm [25] using the $\mathcal{CK}$ local search and different parameter settings for the perturbation size (there 5 *interchange* moves) and the accetpance criterion (there, only better quality solutions

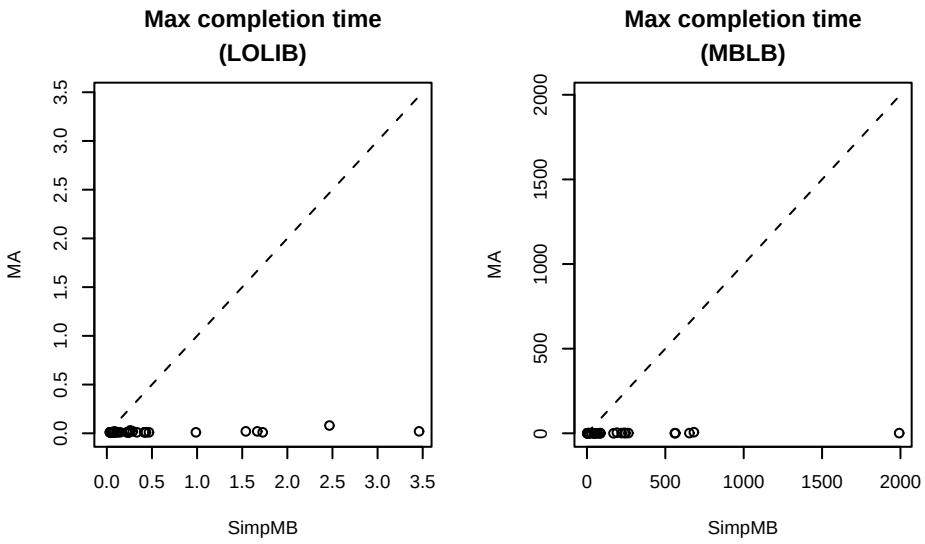

Figure 8: Pairwise comparison of SimpMB (indicated as exact method) to MA for the LOLIB and MBLB instances. For each instance, the time given for the MA is the maximum time over 100 trials to find a globally optimal solution (y-axis), while the timing for SimbMB (given in the x-axis) is obtained from running it once, because it is a deterministic algorithm.

were accepted). Hence, we suspect that the main reason for the "poor" performance of ILS is that the configuration returned from the automatic tuning procedure performs poorly on MBLB instances. Hence, we tested again this earlier ILS implementation, referring to it in the following as ILSv5, with the only difference that now we use the $\mathcal{LS}_f$ local search instead of $\mathcal{CK}$.

In fact, ILSv5 resulted to be far better than ILS; it was able to reach the optimal solution in all trials for every instance, and the computation times were even smaller than that of MA, as Figure 10 shows. The hardest instance was solved by ILSv5 in a maximum computation time of less than 5 seconds, while all other instances took less than one second.

These result suggest, in turn, that ILSv5 may perform better than ILS also on other instance classes. We tested this conjecture on the LMC-LOP instances of dimension 75 and the XLOLIB instances of size 150. For these instances, we computed the average deviation from the best known solutions and plotted these in Figure 11; the results is that ILSv5 is significantly worse than ILS on these instances.

Overall, these results suggest that the performance of MA is more robust with respect to the various instance classes than ILS. This conclusion can be drawn because (i) ILS and MA were only tuned on the dimension 100 instances from XLOLIB, but MA shows, in general, good behavior across all instance classes and (ii) variants of ILS that differ only in some details of the parameter tuning make a large difference to

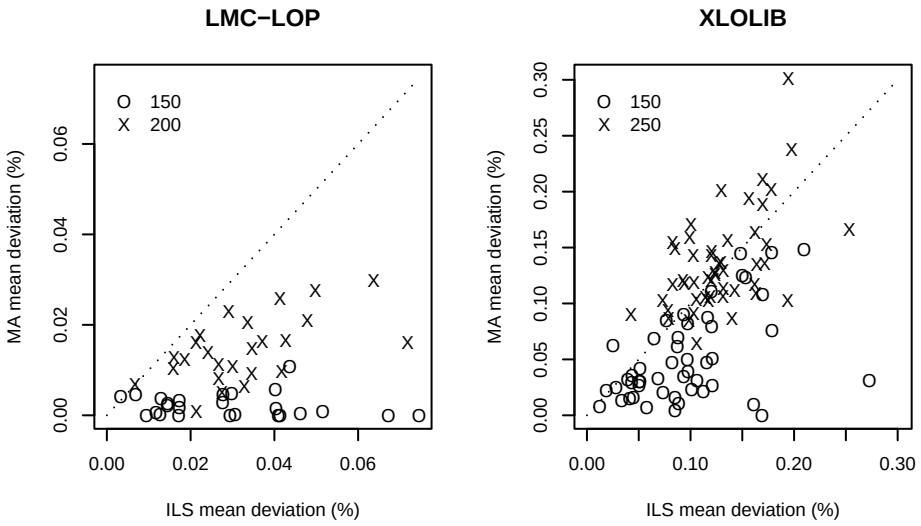

Figure 9: Pairwise comparison of the average solution quality obtained by ILS and MA, on large LMC-LOP instances (left) and XLOLIB instances (right). The results are grouped according to the different instance sizes (indicated by crosses or circles). Each cross (circle) gives on the x-axis the average deviation from the best known solution found by ILS and on the y-axis that of the MA.

ILS performance and, hence, make parameter settings strongly dependent on instance classes.

7.4 FDC and instance hardness

The experiments with ILS on the LMC-LOP instances indicate that ILS has significant difficulties for solving all the instances in all the runs. One reason may be that ILS is attracted to high quality solutions that may be far from a globally optimal one. But, if this is the case, it is likely that ILS shows such a behavior on instances with a low FDC value.

This conjecture is examined by analyzing ILS results in dependence of the FDC coefficient ρ and the easy level FDC coefficient ρ'. As an index of how hard is an instance for ILS, we used the average deviation from the best known solutions and the number of best known solutions returned by ILS over 30 trials. The plots in Figure 12 illustrate graphically the relationship of these measures to the FDC and easy level FDC and show that these search space characteristics affect the hardness of an instance as encountered by ILS. The FDC has a strong negative correlation with the average deviation from the optima found by ILS and a strong positive one with the number of global optima. Summarizing, the higher is the FDC the easier become the instances for ILS, as we conjectured. A slightly stronger correlation is observed for the *easy level*

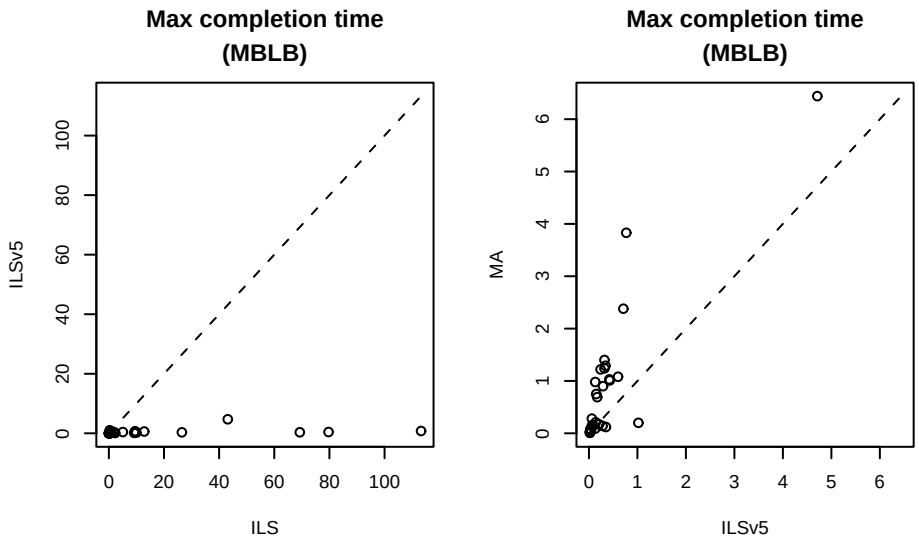

Figure 10: Pairwise comparison of ILSv5 to ILS (left) and ILSv5 to MA (right) based on the maximum time measured across 100 trials to find a globally optimal solution on MBLB instances.

FDC, which may suggest that the easy level FDC is better suited to predict the instance hardness for ILS.

7.5 Comparison

In the literature, we find three main metaheuristic approaches to the LOP. In [7], Campos, Laguna, and Martí proposed the application of Scatter Search (SS) to the LOP and they discussed several ways of how to implement an SS approach to the LOP.[3] The same authors studied an elite tabu search algorithm with additional diversification features for the LOP [17]. In this latter article, they also presented the instance class LMC-LOP. The most recent metaheuristic application for the LOP is the iterated dynasearch (IDS) of Congram [9]. Dynasearch is a local search algorithm where a dynamic programming approach is used to find the best set of independent *insert* moves (two moves are independent if they do not overlap); iterated dynasearch is then simply an ILS algorithm that uses dynasearch in the local search step.

In the following, we give some comparisons on the solution quality and the timings between the different available approaches. However, we encountered several difficulties for doing so. The least severe probably is that the experiments in these articles were run on different machines. Using [29] and some experimental test we evaluated that the machine we use is 21 time faster than the Intel Pentium 166Mhz used in [7, 17]

[3]The scatter search of [6] differs only in minor details from the one in [7] and the results are essentially very similar. Therefore, we focus in the following on the results presented in the first article.

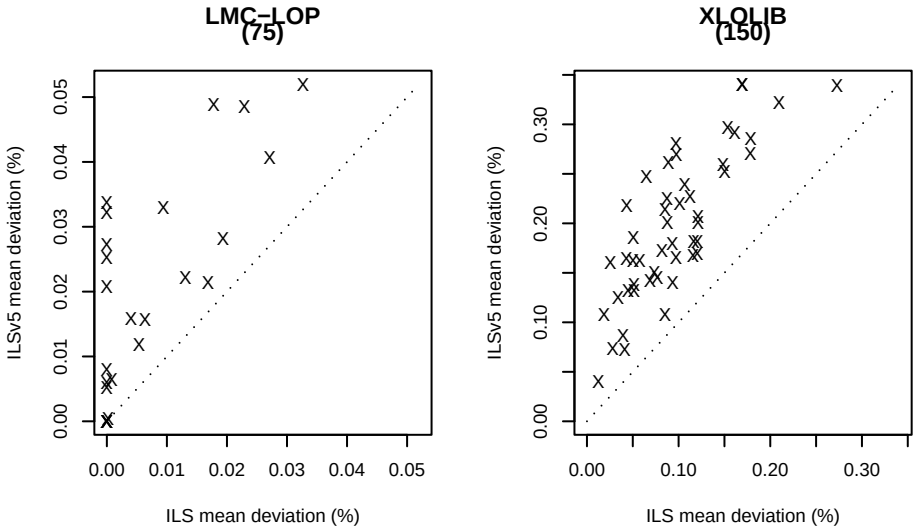

Figure 11: Pairwise comparison between the ILS and ILSv5 on LMC-LOP and XLOLIB instances. Each cross gives on the x-axis the average deviation from the best known solution found by ILS and on the y-axis that of ILSv5.

and 15 times faster than the Power Challenge R10000 used in [9]. In Table 10 we report some results for the instance classes studied in [7, 17, 9]. The major difficulty for the comparison we found were that not enough details were given in these articles to allow a detailed comparison. First, the termination criterion applied to ETS and IDS is not clearly stated, neither how many trials were run on the different available instances. Second, the "average deviation" is the mean of the results over all the instances of the considered class; however is not clear which results are reported (for example, if it is the mean over the experiments or the best results obtained). Because of these problems it is not possible to establish if the number of optima given in these articles is the number of instances for which the algorithm was able to get at least once a global optimum, always the optimum, or the result after just one run. To be on the most cautious side (that is, to let the reported results appear in the best possible light), we will assume that averages for SS, ETS, and IDS are given as averages of the best solutions found and that the number of optima is the number of instances that are always solved to optimality.

For the results of MA and ILS, we report the average deviation computed over all results obtained in 100 runs for LOLIB and 30 runs for LMC-LOP instances. (Note, that for these instances the MA found optimal solutions in each single trial for all the instances of LOLIB and LMC-LOP.) For the time we indicate the average time to find an optimal solution when it was found. As the number of optima we report how many instances were solved to the optimum in all the considered runs by the MA, while for ILS we additionally give the number of instances for which a global optimum was

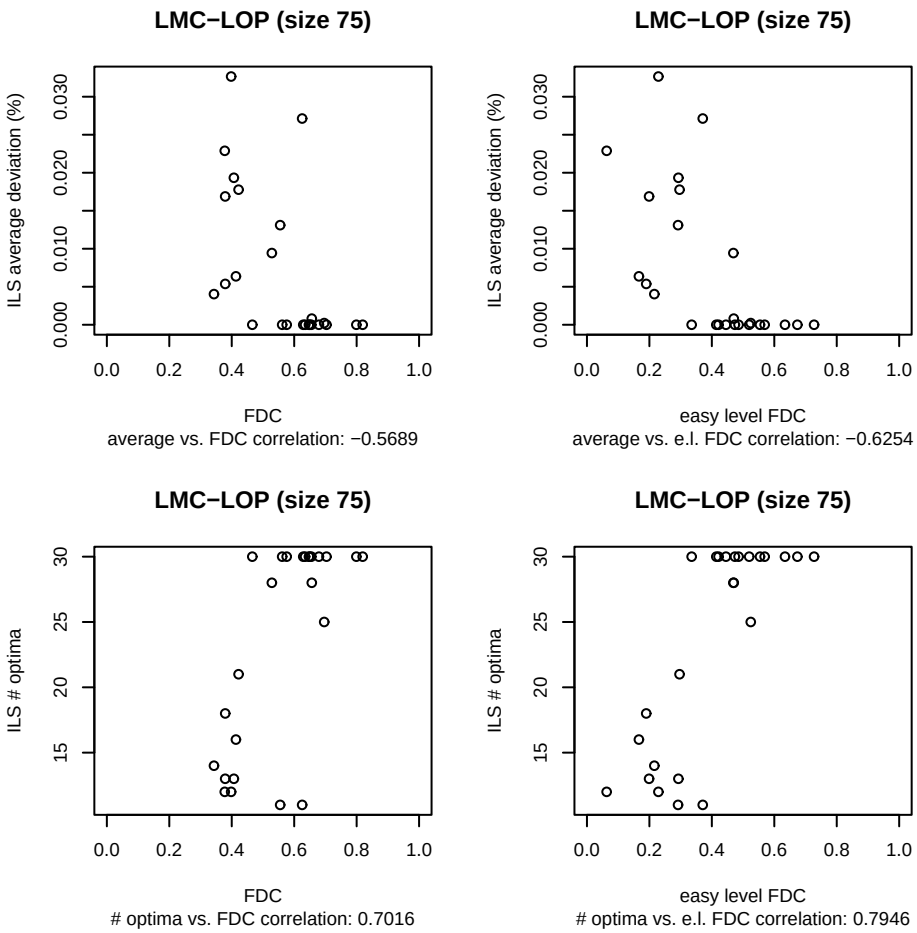

Figure 12: FDC affects the instance hardness. Shown are plots of the average percentage deviation from pseudo-optimal solutions for ILS (in the first row) and the number of best known solutions found by ILS (second row) versus the FDC (left column) and the easy level FDC (right column). In addition, are given the correlations between each pair of measures.

Table 10: Comparison our ILS and MA algorithm to three algorithms from the literature. Avg.Dev.(%) gives the average percentage deviation from the known optimal solutions, # optima the number of optimal solutions found, run time(s) gives the runtimes reported in the original papers, and "P166 run time" are the computation times translated to a 166 Mhz Pentium CPU.

		SS	ETS	IDS	ILS	MA
LOLIB	Avg.Dev.(%)	0.01	0.00	0.00	0.00	0.00
	# optima	42	47	49	49	49
	run time(s)	3.82	0.93	1.22	0.00165	0.00176
	P166 run time	3.82	0.93	0.30	0.35	0.37
LMC-LOP size: 75	Avg. Dev.(%)	–	0.05	0.00	0.0072	0.00
	# optima	–	3	25	13(25)	25
	run time(s)	–	2.95	10.56	6.58*	0.38
	P166 run time	–	2.95	38.25	138.15	7.90

Table 11: Comparison of ILS and MA to ETS on large LMC-LOP instances. Given is the average percentage deviation from the best known solutions, averaged over all trials and all instances.

size	ETS	ILS	MA
150	0.18	0.029	0.0022
200	0.19	0.033	0.015

found at least once.

Regarding the results in Table 10, let us remark the following. In [9] is reported that the IDS was able to obtain always the same best result for the LMC-LOP instances. Therefore, we assumed that this is the very same result we obtained (and, hence, the resulting average deviation of 0.0 for IDS in Table 10 from the best known solution). Instead, for ETS we can give precise results, because Dr. Rafael Martí send us a spreadsheet containing the results of ETS for each instance.

When comparing the results in Table 10, it is clear that, even under the cautious assumptions about the results of SS, ETS, and IDS, our ILS and MA algorithms are extremely competitive to the earlier proposed metaheuristic approaches to the LOP. In fact, ILS and MA outperform ETS and SS on the LOLIB instances and are roughly on par with IDS. On the small LMC-LOP instances, ILS and MA return much better quality solutions than ETS at, however, higher computation times. Our ILS appears to perform slightly worse than IDS on these instances, while MA solves the small LMC-LOP instances about five times faster than IDS.

Finally, we compares the average deviation from the best known solutions for the LMC-LOP instances of dimension 150 or 200 for ETS, ILS, and MA in Table 11. (Note that ETS results were adjusted to the new best known solutions for these instances.) The results show that ILS and MA yield by far better quality solutions than ETS; however, it is not clear how the computation times of the three algorithms compare, because not enough details are given in [17].

The overall result of the comparison is that, in particular, the MA obtains an ex-

tremely encouraging performance both, from the point of view of the time and the solution quality reached. In fact, even when being cautious about the experimental conditions used in the other papers, our results suggest that the MA is a new, very robust state-of-the-art algorithm for the LOP.

8 Conclusions

In this paper we have given a detailed analysis of benchmark instances for the LOP. These include a new class of instances, called XLOLIB.[4] The instances of this class are randomly generated through sampling real-world instances, which allows to derive large, random real-world like instances. In fact, cross-statistical data on the distribution of the matrix entries of XLOLIB instances are basically the same as those of the underlying real-world instances from LOLIB. However, the search space analysis showed some discrepancy between the original LOLIB instances and the newly generated XLOLIB instances. Nevertheless, XLOLIB instances appear to be much closer to real-world instances than instances from other, randomly generated classes like MBLB or LMC-LOP instances.

The search space analysis of LOP instances showed that most instances have high correlation length, suggesting that, in general, the LOP is easy to solve when compared to other problems [2]. Furthermore, most LOP instances have also a high fitness distance correlation. Notable exceptions occur for a few LOLIB instances, where even negative fitness distance correlations were found. Concerning measures of search space characteristics, we introduced a new way to measure the FDC, which we named "easy level FDC". This measure tries to consider the fact that metaheuristics actually search through high quality local optima and the central idea of the easy level FDC is to focus the analysis on high quality solutions. In fact, the easy level FDC showed a better correlation to the instance hardness for of small LMC-LOP instances for ILS algorithms than the standard way of determining FDC.

Based on the results of the search space analysis and the high solution quality returned by simple iterative improvement algorithms, we further studied efficient iterated local search and memetic algorithms for the LOP. A comparison between the two algorithmic approaches showed that the MA resulted in a much more robust performance with respect to the different instance classes than the ILS algorithm. However, some additional experiments have shown that the ILS algorithm is more sensible to parameter settings and that the performance of different variants depends strongly on the instance classes. It is an open question whether, with appropriate tuning, ILS can reach MA's performance on all the instance classes. A final comparison of MA and ILS performance to other available metaheuristic approaches for the LOP showed that our MA is a new, very robust state-of-the-art algorithm for the LOP.

[4] All the instances and the best known results will be published on the WWW at the address `http://intellektik.informatik.tu-darmstadt.de/~schiavin/lop` for their easy access.

Acknowledgments

The authors would wish to thank Prof. John Mitchell and Dr. Brian Borchers for making available the code of their exact algorithm. We thank also Dr. Rafael Martí for sending to us the LMC-LOP instances and the results they obtained with the Tabu Search.

This work was supported 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] E. Angel and V. Zissimopoulos. Autocorrelation coefficient for the graph bipartitioning problem. *Theoretical Computer Science*, 191:229–243, 1998.

[2] E. Angel and V. Zissimopoulos. On the classification of NP-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99:261–277, 2000.

[3] O. Becker. Das Helmstädtersche Reihenfolgeproblem – die Effizienz verschiedener Näherungsverfahren. In *Computer uses in the Social Science*, Wien, January 1967.

[4] M. Birattari, Thomas Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

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

[6] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, 21(4):397–414, 2001.

[7] V. Campos, M. Laguna, and R. Martí. Scatter search for the linear ordering problem. In D. Corne et al., editor, *New Ideas in Optimization*, pages 331–339. McGraw-Hill, 1999.

[8] S. Chanas and P. Kobylanski. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6:191–205, 1996.

[9] R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.

[10] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.

[11] M. Grötschel, M. Jünger, and G. Reinelt. Optimal triangulation of large real world input–output matrices. *Statistische Hefte*, 25:261–295, 1984.

[12] H.H. Hoos and T. Stützle. Evaluating Las Vegas algorithms — pitfalls and remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann, San Francisco, 1998.

[13] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L.J. Eshelman, editor, *Proc. of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufman, San Francisco, 1995.

[14] R. Kaas. A branch and bound algorithm for the acyclic subgraph problem. *European Journal of Operational Research*, 8:355–362, 1981.

[15] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[16] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison Wesley, New York, 1993.

[17] M. Laguna, R. Martí, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers & Operations Research*, 26(12):1217–1230, 1999.

[18] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, Norwell, MA, 2002.

[19] P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000, 2000.

[20] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw-Hill, London, 1999.

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

[22] J. E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. In H. L. Frenk *et al.*, editor, *High Performance Optimization*, pages 349–366. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[23] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 105–144. Kluwer Academic Publishers, Norwell, MA, 2002.

[24] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operational Research*, 86:473–490, 1999.

[25] T. Schiavinotto and T. Stützle. Search space analysis of the linear ordering problem. In G. R. Raidl et al, editor, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 322–333. Springer Verlag, Berlin, Germany, 2003.

[26] P. Stadler. Towards a theory of landscapes. In R. Lopéz-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertuche, editors, *Complex Systems and Binary Networks*, volume 461, pages 77–163, Berlin, New York, 1995. Springer Verlag.

[27] P. Stadler. Landscapes and their correlation functions. *J. of Math. Chemistry*, 20:1–45, 1996.

[28] P. F. Stadler and W. Schnabl. The landscape of the travelling salesman problem. *Physics Letters A*, 161:337–344, 1992.

[29] Standard Performance Evaluation Corporation. SPEC CPU95 and CPU2000 Benchmarks. `http://www.spec.org/`, November 2002.

[30] T. Stützle and H. H. Hoos. Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, Operations Research/Computer Science Interfaces Series, pages 589–611. Kluwer Academic Publishers, Boston, MA, 2001.

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

[32] G. Syswerda. Schedule optimization using genetic algorithms. In L. Davis, editor, *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1990.

[33] E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.