Search Space Analysis of the Linear Ordering Problem

Tommaso Schiavinotto and Thomas Stützle

Darmstadt University of Technology, Intellectics Group,
Alexanderstr. 10, 64283 Darmstadt, Germany
{schiavin,tom}@intellektik.informatik.tu-darmstadt.de

Abstract. The Linear Ordering Problem (LOP) is an $\mathcal{NP}$-hard combinatorial optimization problem that arises in a variety of applications and several algorithmic approaches to its solution have been proposed. However, few details are known about the search space characteristics of LOP instances. In this article we develop a detailed study of the LOP search space. The results indicate that, in general, LOP instances show high fitness-distance correlations and large autocorrelation length. However, there exist significant differences between real-life and randomly generated LOP instances. Based on these observations the the limited size of real-world instances, we propose new, large real-life like LOP instances which appear to be much harder than other randomly generated instances. Additionally, we propose a rather straightforward Iterated Local Search algorithm, which shows better performance than several state-of-the-art heuristics.

1 Introduction

The Linear Ordering Problem (LOP) is an $\mathcal{NP}$*–hard* problem that has a large number of applications in such diverse fields as economy, sociology, graph theory, archaeology, and task scheduling [6]. Given an $n \times n$ matrix C, the 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 of matrix C such that the sum of the elements in the upper triangle is maximized.

Both exact algorithms [6, 4, 14] and approximate algorithms have been proposed for this problem. State-of-the-art exact algorithms can solve fairly large instances with up to a few hundred columns; however, their computation time increases strongly with instance size. Approximate algorithms include constructive algorithms like Becker's greedy algorithm [1], local search algorithms, the $\mathcal{CK}$ heuristic [3], as well as metaheuristics such as Elite Tabu Search [10], Scatter Search [2], or iterated Dynasearch [5].

These algorithms have typically been tested on real-world as well as randomly generated instances. The LOLIB benchmark library comprises 49 real-world instances that are input-output tables of economical flows in the EU. This is the most widely used set of instances to test algorithms for the LOP, and it is available from [16]; for all LOLIB instances optimal solutions are known [7]. Because LOLIB instances are rather small, Mitchell and Borchers [14] generated larger instances in their research on exact algorithms for the LOP. The idea underlying their way of generating instances is that there should be a large number of solutions with costs close to the optimal value to obtained hard instances. Thirty of these instances with known optimal solutions are available from [13], where also the generator can be found; we will refer to this instances as MBLB (Mitchell-Borchers LOP Benchmarks). Other instances were based on random generators [10], however, the original instances are not publically available and therefore not used in this paper.

One of the contributions of this paper is an analysis of the search space characteristics of the available LOP instances including an autocorrelation analysis and a fitness-distance analysis. The results indicate significant differences between the LOLIB and MBLB instances, surprisingly suggesting that MBLB instances should be, when adjusting for the difference in size, easier to solve for metaheuristics. We generated randomly large real-life like instances and computational results confirm our conjecture. Additionally, our results of the search space analysis suggest that Iterated Local Search (ILS) algorithms [11] are likely to achieve high performance for the LOP. In fact, a rather straightforward ILS algorithm using the $\mathcal{CK}$ local search heuristic appears to be competitive or superior to all previously proposed metaheuristic approaches.

The paper is structured as follows. In the next section we analyze the behavior of different local search algorithms on the two instance classes. Sections 3 and 4 give statistical measures on the instance structure and the results of the landscape analysis. Finally, we give details on the performance of two Iterated Local Search implementations based on two different local searches and give some concluding remarks in Section 6.

2 Local Search

The currently best known constructive algorithm, due to Becker [1] orders the columns (and the rows) based on a heuristic value

$$q_j = \frac{\sum_{k=1}^{m} c_{jk}}{\sum_{k=1}^{m} c_{kj}}.$$

The higher q_j the sooner the column with index j must be in the permutation. This algorithm runs in $\mathcal{O}(n^3)$ is fast and it gives good solutions that can be used as starting point for local search algorithms. The average deviation from the optimum solutions for LOLIB instances with Becker's heuristic is 9.46% (compared to an average of 30.48% for random permutations) and 2.38% on MBLB instances (random permutations average 40.34% above optimum).

Better solutions are obtained using local search algorithms. We run some initial experiments with a number of iterative improvement algorithms. We considered two different neighbourhoods, which are defined by the operations applicable to a current solution. The first is 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 $i \neq j$:

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

We denote this neighbourhood by $\mathcal{N}_I$; its size is $|\mathcal{N}_I| = n^2 - 2n$.

Another possible neighbourhood is $\mathcal{N}_X$, defined by the operation *interchange*, where to elements are exchanged. *Interchange* : $\Pi \times \{1,\ldots,n\}^2 \rightarrow \Pi$, for $i \neq j$:

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

The size of this neighbourhood is $|\mathcal{N}_X| = \frac{n(n-1)}{2}$. We implemented both, a best- and first-improvement version of local search based on the $\mathcal{N}_I$ and the $\mathcal{N}_X$ neighbourhoods. In addition we also used a *variable neighbourhood descent* (VND) style heuristic: if a local minimum with respect to the first neighbourhood is met, the search is continued exploiting the second neighbourhood.

In addiction to these standard neighbourhoods, we also implemented a LS algorithm $\mathcal{CK}$ due to Chanas and Kobylański [3] that uses two functions *sort* and *reverse*. When applied

Table 1. Comparison of first-improvement algorithms based on different neighbourhoods on MBLB and LOLIB instances (B.I. indicates that a best-improvement local search was applied). The results are averaged over 100 runs on each of the 30 instances, the number of optima indicates on how many instances was found a global optimum at least once in the 100 runs, and the time given is the total time spent to run the local search on all 30 instances once. If two neighborhoods are indicated (separated by $+$), VND is used; in this case we give the percentage of how often and for how many instances the local search in the second neighbourhood was able to improve the first one.

	Local Search	Avg. Dev. (%)	# Opt.	time (s)	# Inst. Improved	% Improved Runs
MBLB	$\mathcal{N}_I$	0.0195	10	10.04	–	–
	$\mathcal{N}_I$ (B.I.)	0.0219	11	187.72	–	–
	$\mathcal{N}_X$	0.3870	0	9.35	–	–
	$\mathcal{CK}$	0.0209	12	**0.22**	–	–
	$\mathcal{N}_X + \mathcal{N}_I$	0.0182	11	23.33	**30**	**100.00**
	$\mathcal{N}_I + \mathcal{N}_X$	0.0191	**14**	11.44	0	0
	$\mathcal{N}_I + \mathcal{CK}$	**0.0169**	10	9.94	26	26.30
	$\mathcal{CK} + \mathcal{N}_I$	0.0197	**14**	0.90	3	0.40
LOLIB	$\mathcal{N}_I$	0.1842	42	0.1802	–	–
	$\mathcal{CK}$	0.2403	38	**0.0205**	–	–
	$\mathcal{N}_I$+$\mathcal{CK}$	**0.1819**	**44**	0.1881	**33**	**8.37**
	$\mathcal{CK}$ +$\mathcal{N}_I$	0.2360	40	0.0420	2	0.08

to a permutation, *sort* returns a new one in which the elements are rearranged according to a sorting criteria similar to the one used in Becker's algorithm, while *reverse* returns simply 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:

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

where the $\circ$ is the 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:

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

Unfortunately the $\mathcal{CK}$ local search induces an ill-defined neighbourhood that cannot be used for all the types of search space analysis conducted in the next section.

Table 1 shows the results obtained by applying the various LS algorithms to random initial solutions. It can be noticed that $\mathcal{CK}$ is by more than an order of magnitude faster than the other single neighbourhood local searches and it obtains a solution quality comparable to the other local searches. The best solution quality among plain local searches is obtained with the $\mathcal{N}_I$ neighbourhood, while local search in the $\mathcal{N}_X$ neighbourhood returns poor quality solutions using comparable computation times to *insert*. The VND algorithms shows that only by concatenating the $\mathcal{N}_I$ with $\mathcal{CK}$ yields significant improvements over the plain local search, while it is not the case vice versa. In general, these preliminary experiments suggest that good results can be obtained using a single local search and that the improvement through VND is not important (besides the case of *interchange+insert*). Hence using a plain local search allows a simpler design of a metaheuristic and a shorter computation time. In the rest of this work we will focus on *insert* and $\mathcal{CK}$, because they showed to be the best performing in terms of quality.

Table 2. LOLIB (left) and MBLB (right) structural information: some statistical information is given about the structure of the instances, the 1st and 3rd Qu. indicate the first and the third quantile respectively.

Size		Min	1st Qu.	Median	3rd Qu.	Max	Mean
44	Sparsity	0.00	21.53	41.27	55.06	80.63	39.36
	VC	4.21	4.59	5.13	5.53	10.34	5.49
	Skewness	9.78	11.51	13.01	15.31	25.70	14.28
50	Sparsity	33.65	35.65	43.16	52.08	58.30	44.57
	VC	5.54	5.72	6.96	10.14	16.17	8.91
	Skewness	13.13	15.16	21.09	29.55	39.21	23.63
56	Sparsity	25.51	25.95	26.91	27.18	28.06	26.67
	VC	4.24	4.38	4.45	5.01	6.45	4.85
	Skewness	10.84	11.48	12.24	16.81	30.52	15.67
60	Sparsity	28.69	28.78	28.86	29.42	29.97	29.17
	VC	5.85	5.98	6.11	6.12	6.14	6.03
	Skewness	23.84	23.96	24.09	24.47	24.85	24.26

Variation Coefficient

Size	Sparsity		
	0%	10%	20%
100	-	-	1.00-1.02
150	0.77-0.78	0.88-0.89	-
200	0.77-0.78	0.88	-
250	0.77-0.78	-	-

Skewness

Size	Sparsity		
	0%	10%	20%
100	-	-	0.98-1.00
150	0.82-0.85	0.87-0.88	-
200	0.82-0.85	0.88-0.89	-
250	0.82-0.84	-	-

3 Structural analysis of the instances

The first level of our analysis of LOP instances is based exclusively on a high level description of the instances and on the input data, that is, the distribution of the matrix entries.

LOLIB comprises 49 real world instances, of which 39 are of size $n = 44$, 5 of size $n = 50$, 11 of size $n = 56$, and 3 of size $n = 60$. MBLB instances are randomly generated, with the matrix entries generated according to a uniform distribution, and then certain number of zeros are added. MBLB comprises 30 instances: 5 instances of size 100, 10 of size 150, 10 of size 200 and 5 of size 250.

For all instances we computed the sparsity, the variation coefficient, and the skewness of the matrix entries. The sparsity measures the percentage of elements that are equal to zero; it seemed to have a strong influence on algorithm behavior in [14]. The sparsity of the real world examples varies very strongly from 0% to 80%, while MBLB instances where generated with fixed sparsity of 0%, 10%, and 20%. Two further measures, which depend to some extent on the sparsity, were computed. The *variation coefficient* (VC) is defined as $\frac{\sigma}{\bar{X}}$, where σ is the standard deviation and $\bar{X}$ is the mean of the matrix entries: it gives an estimate of the variability independent of the size and the range of the matrix entries. The *skewness* is the third moment of the mean normalized by the standard deviation, it indicates the degree of asymmetry of the matrix entries.

Table 2 gives some information about these measures for LOLIB and MBLB instances. We present the results for the two instance classes in two different ways, because of the low variance of these values on the MBLB instances and because these instances can easily be grouped on sparsity. As we see, the latter measure determines the skewness and the VC for MBLB instances. This analysis already shows that LOLIB and MBLB instances are very different. While MBLB instances are rather similar (see the low variation in our measures) and generally have low VC and skewness, the real-world instances show much larger differences in all three statistics and the VC and skewness are typically much larger.

4 Landscape analysis

Landscape analysis is an instrument used in order to study not trivial features of combinatorial problems, which can be traced back to [21]. The idea is to "visualize" the search space as a landscape formed by all the solutions (in our case permutations) and a *fitness* value for each solution corresponding in our case to the objective function f.

Table 3. Given is the value $\frac{\ell}{n}$, that is the correlation length normalized by instance size for LOLIB and MBLB instances.

	Size	Min	1st Qu.	Median	3rd Qu.	Max	Mean
LOLIB	44	0.7536	0.7854	0.7929	0.8031	0.8237	0.7937
	50	0.7642	0.7695	0.7861	0.8043	0.8145	0.7877
	56	0.8004	0.8120	0.8241	0.8311	0.8371	0.8208
	60	0.8383	0.8389	0.8395	0.8399	0.8403	0.8394
MBLB	100	0.9339	0.9350	0.9357	0.9360	0.9371	0.9355
	150	0.9594	0.9595	0.9610	0.9623	0.9645	0.9612
	200	0.9626	0.9690	0.9696	0.9710	0.9744	0.9698
	250	0.9703	0.9742	0.9748	0.9769	0.9775	0.9747

Formally, a landscape for 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. If we think of a local search as the operation to find a local optimum, it is natural to define distance according to the type of neighborhood we use. In this work we focus on the *insert* and $\mathcal{CK}$ neighborhoods. For each neighborhood, the distance is defined as the minimum number of applications of the basic local search operation to transform a permutation into another. While for the $\mathcal{CK}$ algorithm it is not clear which is this *basic operation* is, in the case of the *insert* local search it is clearly the *insert* operation. Unfortunately, as far as we know, it seems that there is no efficient way to compute the number of minimum *insert* applications, in order to transform a permutation into another one. Therefore, we use a surrogate distance that evaluates the distance between two permutation considering the position of the elements in both, originally proposed in [15]. We call this distance *precedence metric*: for each the element j it is counted the number of times this is preceded by another element i in both permutations. The final 'distance' is obtained subtracting this quantity from $n(n-1)/2$. Hence, the maximum distance between two sequences $n(n-1)/2$, this value corresponds also to the *diameter* of the landscape.

One of the features of a fitness landscape often investigated is the ruggedness: a fitness landscape is said to be rugged if there is a low correlation between neighbouring points. In order to calculate this correlation [18, 17, 20] suggest to use the autocorrelation on a time series of "adjacent" solutions that are generated by a *random walk*:

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

This measure is defined on a time series $\{f(x_t)\}$ and defines the correlation of two solutions s steps away along a random walk of length m through the fitness landscape ($\sigma^2(f)$ is the variance of the time series, and $\bar{f}$ the mean). Another measure, based on $r(s)$, simpler to interpret, is the *landscape correlation length* $\ell = -\frac{1}{ln(|r(1)|)}$ $(r(1) \neq 0)$: the lower is the value for ℓ, the more rugged is the landscape. We computed ℓ on the LOLIB and MBLB instances based on the $\mathcal{N}_I$ neighbourhood, because the $\mathcal{CK}$, as said before, can not be used for this type of analysis. Table 3 summarizes data collected all over the instances grouped by size. The correlation length is always smaller for LOLIB instances than for MBLB instances, which, abstracting from the instance size, indicates that the LOLIB problems are harder for *insert* than the MBLB ones.

The next step in the our analysis was to generate a large number of local optima for all the instances (13,000 for LOLIB and 1,000 for MBLB instances). Based on these, we first

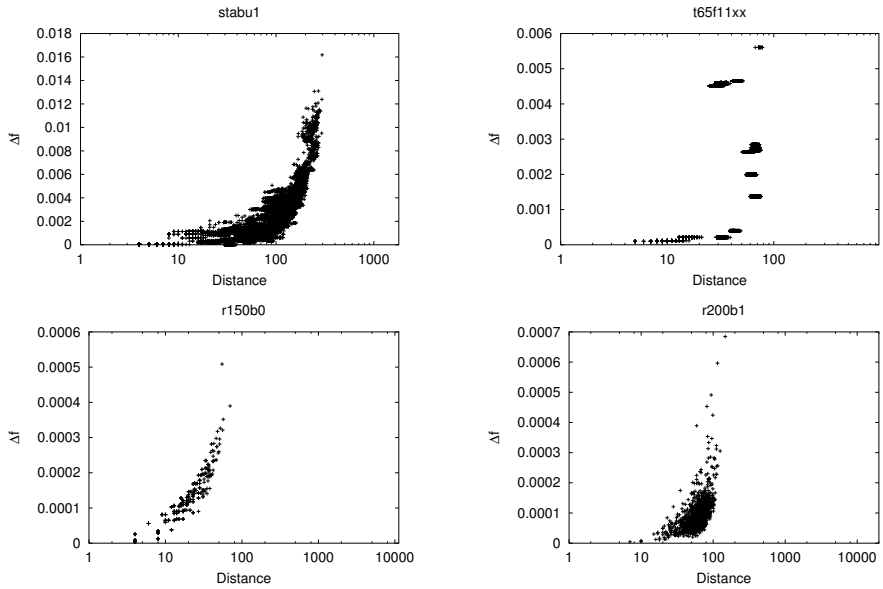

Fig. 1. Plots of the deviation from the optimal value versus the minimum distance of local optima (generated with *insert* LS) from a global optimum, the maximum distance is shown. From LOLIB: `stabu1` $\rho = 0.8757$; `t65f11xx` $\rho = -0.1056$ (lowest). From MBLB (log scale on distance): `r150b0` $\rho = 0.9395$ (highest); `r200b1` $\rho = 0.6144$ (lowest).

analyzed the number of global optima that were found among the local optima. Among the local optima generated by the *insert* LS, we find on average 0.46% distinct global optima over the distinct l.o. for MBLB (min. 0%, max. 4.27%) and 14.10% for LOLIB (min 0.47%, max 85.12%); this indicates that most of the instances can actually be solved by a random restart algorithm that is run long enough. Furthermore, as Table 1 shows, local optima for MBLB instances have a smaller deviation from the optimal value than for LOLIB instances.

Another standard technique in analyzing the landscape is the study of the fitness-distance correlation (ρ) between the quality of the local optima and the distance from the closest global optima [8]. Here we focus on the relationship between the distance to the closest globally optimal solution and the deviation from the global optimum as the fitness function. In this case, a strongly positive ρ coefficient, $-1 \leq \rho \leq 1$, indicates that the solution quality gives good guidance when searching for global optima (the interpretation then is that the smaller the deviation, that is, the better the solution, the closer we get to global optima, on average). Tab. 4 summarizes the information about fitness/distance correlation, Fig. 1 gives some example plots of the fitness-distance relationship.

All MBLB instances and most LOLIB instances show a very high value for ρ, suggesting that these instances should be relatively easy for restart type algorithms [12]. However, some of the LOLIB instances show even negative fitness-distance correlation; therefore, these instances may pose some problems to metaheuristics despite their small size. In general, the variability of ρ is much higher for LOLIB instances, suggesting that these show a variety of different structures, different from MBLB instances, which are more similar to each other. Additionally, ρ is typically smaller for LOLIB instances than MBLB instances, suggesting that they should be somewhat harder than MBLB instances. Finally, we run a

Table 4. Statistical information about fitness/distance correlation (ρ) on local optima returned by $\mathcal{CK}$ (left table) and *insert* (right table) for both classes of instances grouped by size.

	Size	Min	1st Qu.	Median	3rd Qu.	Max	Mean
LOLIB	44	-0.07	0.48	0.67	0.83	1.00	0.59
	50	0.17	0.26	0.49	0.74	0.92	0.52
	56	0.34	0.67	0.72	0.85	0.92	0.73
	60	0.67	0.73	0.79	0.84	0.88	0.78
MBLB	100	0.68	0.75	0.75	0.77	0.83	0.76
	150	0.72	0.84	0.84	0.87	0.90	0.84
	200	0.53	0.66	0.75	0.77	0.91	0.73
	250	0.79	0.79	0.85	0.86	0.87	0.83

	Size	Min	1st Qu.	Median	3rd Qu.	Max	Mean
LOLIB	44	-0.11	0.42	0.64	0.76	1.00	0.58
	50	-0.01	0.40	0.61	0.75	0.89	0.53
	56	0.34	0.65	0.70	0.85	0.94	0.72
	60	0.70	0.71	0.73	0.80	0.88	0.77
MBLB	100	0.71	0.71	0.73	0.81	0.84	0.76
	150	0.70	0.79	0.84	0.86	0.94	0.82
	200	0.61	0.71	0.73	0.78	0.91	0.75
	250	0.79	0.80	0.82	0.84	0.86	0.82

Table 5. Summary of the maximum time for finding the optima over 100 runs (the optima has been found in every run) of ILS$_{\mathcal{CK}}$.

	Size	Min	1st Qu.	Median	3rd Qu.	Max	Mean
LOLIB	44	0.00	0.00	0.01	0.03	0.16	0.03
	50	0.03	0.03	0.09	0.50	1.57	0.44
	56	0.01	0.02	0.03	0.07	0.15	0.05
	60	0.09	0.13	0.16	0.25	0.34	0.20
MBLB	100	0.14	0.20	0.54	0.70	2.41	0.80
	150	0.16	0.21	0.34	0.76	1.81	0.59
	200	0.27	0.94	1.69	4.00	14.27	3.30
	250	0.55	1.44	2.53	3.23	6.25	2.80

	Size	Min	1st Qu.	Median	3rd Qu.	Max	Mean
LOLIB	44	0.02	0.07	0.15	1.19	45.19	3.73
	50	0.16	0.48	6.56	253.90	977.96	247.80
	56	0.160	0.39	0.88	4.66	22.71	4.96
	60	1.40	1.85	2.30	8.12	13.93	5.88
MBLB	100	1.29	1.39	12.37	13.81	34.39	12.65
	150	2.66	3.17	6.94	12.99	27.13	9.27
	200	9.94	21.42	41.60	101.70	158.30	62.51
	250	27.60	49.66	52.21	118.70	133.00	76.23

paired t-test for each class to compare the ρ values obtained by the $\mathcal{CK}$ and *insert* LS on each instance; the result was that they are not significantly different both for LOLIB and for MBLB.

5 Iterated Local Search

Here we consider the application of ILS to the LOP. One main motivation for doing this is the high fitness-distance correlation observed for a large number of LOP instances, which suggests that ILS can successfully solve the LOP. ILS is a metaheuristic that despite its simplicity achieved excellent results on several $\mathcal{NP}$-hard problems [11]. ILS iterates in a particular way over the local search process. This is done based on three main steps: (i) perturb a locally optimal solution, then (ii) locally optimize it with the local search chosen and finally (iii) choose, based on some acceptance criterion, the solution that undergoes the next perturbation phase. In both the implementations we present here, the perturbation is made on $\mathcal{N}_X$. Actually, the only difference between the two implementations is that one is based on $\mathcal{CK}$ LS (ILS$_{\mathcal{CK}}$) and the other on the *insert* LS (ILS$_I$). For both, the parameter tuning has been done on the MBLB instances, for ILS$_{\mathcal{CK}}$ the same settings were good also for LOLIB instances, while the ILS$_I$ exhibits some lack of tuning on the latter class of instances, due to the bias cause by MBLB instances.

We run both algorithms 100 times on all LOLIB and MBLB instances on an AMD Athlon 1.2Ghz machine with 1GB RAM. Table 5 summarizes the maximum time needed to find the known global optima for both instance classes and both ILS algorithms. The results suggest that ILS$_{\mathcal{CK}}$ yields much better performance than ILS$_I$ over the whole benchmark set. For example, ILS$_{\mathcal{CK}}$ was able to find always a global optimum for LOLIB instances in less than 0.34s, except for instance `be75np` where the maximum time for finding a global optimum was of 1.57s. On the MBLB instances ILS$_{\mathcal{CK}}$ takes 14.27s for the `r200e1`, and less than the half for all the others. ILS$_I$ requires much larger computation times. For example, the maximum computation times over 100 runs range between 1.29s and 158.33s for the MBLB, and between 0.02s and 45.19s for LOLIB, with a peak of 978s for instance `be75np` (such a behavior can be in part due to parameter under-tuning).

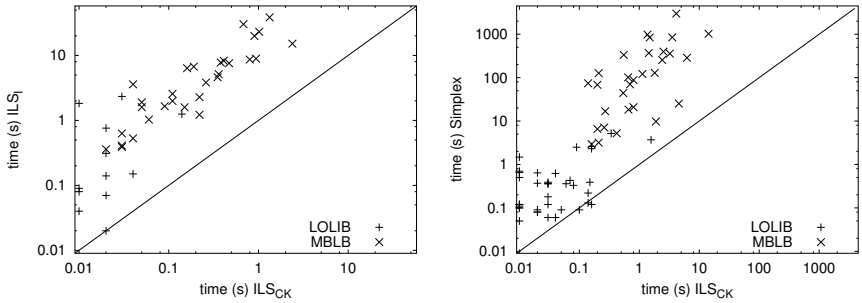

Fig. 2. Pairwise comparison of ILS$_{\mathcal{CK}}$ with ILS$_I$ (the median of the 100 runs is shown) and the Simplex algorithm.

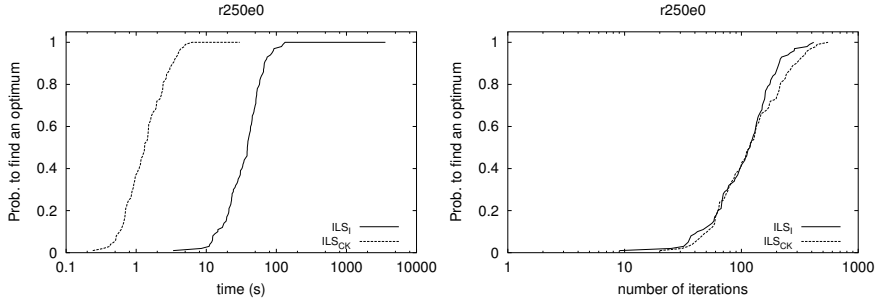

Fig. 3. RTD plots on a MBLB instance: over time (left); over number of iterations (right).

In addition, we run a simplex algorithm by Mitchell and Borchers [14] to compare ILS to exact algorithms. Figure 2 gives a pairwise comparison between ILS$_{\mathcal{CK}}$ and ILS$_I$ (left side) and the Simplex and ILS$_{\mathcal{CK}}$; each point gives the timings of the two different algorithm for the same instance. It must be considered that the comparison with the simplex is somehow inaccurate: this is an exact algorithm, while the ILS does not assure that the optima will be found. In any case, ILS$_{\mathcal{CK}}$ is clearly faster than Simplex, often by several orders of magnitude, and also beats ILS$_I$ on an instance by instance basis.

Runtime distribution (RTD) plots depict the probability of finding a global optima in dependence of computation time. This kind of plots (e.g. Figure 3(a)) point towards an interesting property of the two implementations on MBLB instances: the curves show a similar shape. This suggested to do the same plots but over the number of iterations instead of computation time for the two ILS (Figure 3(b)). As can be observed, by considering only the number of iterations the two algorithms are very similar, and sometimes ILS$_I$ even needs less iterations than ILS$_{\mathcal{CK}}$. In the case shown in Figure 3 ILS$_I$ needs a maximum of 132.99s against 6.25s of ILS$_{\mathcal{CK}}$, but only a maximum of 416 iterations, while ILS$_{\mathcal{CK}}$ needs 552. This indicates that, at least on MBLB instances, the two local searches mainly differ on the speed, and they have not an intrinsically different behavior.

We also compared ILS$_{\mathcal{CK}}$ to published results for existing LOP algorithms (Table 6). The algorithms include a scatter search [2] (SS), elite tabu-search (ETS) [10] and ILS with Dynasearch local search [5]. These algorithms were evaluated on LOLIB instances, instances randomly extracted form the Stanford Graphbase [9], and some additional random instances. Here we focus on LOLIB instances, since we did not have available the other instances. SS and ETS were run on a Pentium 166Mhz and Iterated Dynasearch and a Sun-Sparc 5/110. We estimated (using some indirect comparison between cpu95 and cpu2000 benchmarks found in [19]) that our machine is roughly 15 times faster than the Pentium

Table 6. Comparison of our algorithms with state-of-the-art algorithms, timings for ETS, SS and Dynasearch correspond to the time measured on a Pentium 166Mhz (which is roughly 15 times slower than our machine).

	$ILS_{\mathcal{CK}}$	ILS_I	ETS[10]	SS[2]	Dynasearch [5]
Std. Dev.(%)	0.00	0.00	0.00	0.01	0.00
# Optima	49	49	47	43	49
Avg. Time	0.08	24.06	0.93	3.82	1.22(0.30)

166Mhz. In Table 6 we use the original timings given in the papers. There is clearly a ceiling effect due to the small size of the instances, further experiment on some larger and harder instances could give a better idea of the quality of the algorithms, by the way it seems that our ILS is definitely competitive or may prove to be slightly superior to these best known algorithms for the LOP.

Since the several measures we considered seemed to suggest that LOLIB instances are harder than the MBLB ones we generated new instances of size 250 (as the largest ones in MBLB), bootstrapping the elements from the instances of LOLIB (for each instance a corresponding instance of size 250 has been generated), so that the statistical features were the same. First of all, we calculated the landscape correlation length, and we got that it ranges between 69% and 75% of the size, so a lower value than the original instances. We chose an instance (`be75np`) that resulted to be hard in the original problems. The simplex was not finished after 19000 (about 13 days and 4 hours) minutes, on the MBLB instances the maximum time was 2987.89 seconds. While the $ILS_{\mathcal{CK}}$ was not able to find any optimum over 100 runs of 40 minutes, when on MBLB instances it always found a global optima in all 100 runs in less than 15 seconds.[1]

6 Conclusions

We can draw several conclusions from this paper. First, LOLIB and MBLB instances are significantly different, showing different high-level characteristics of the matrix entries like sparsity and skewness. Second, these differences also show up in the results of a search space analysis, in which we found that MBLB instances typically have higher correlation length and also a generally larger fitness-distance correlation than LOLIB instances. This suggests that MBLB instances, should be easier to solve than LOLIB instances, when abstracting from instance size. Third, we developed a new state-of-the-art ILS algorithm, which is able to find global optima solutions to all benchmark instances in short computation time. Finally, we generated new, large real-life like instances, which appear to be significantly harder to solve than MBLB instances of the same size.

In future work we will extend the search space analysis to a larger number of of instances, including all the different types of instances proposed so far in the literature and examine more closely the relationship of sparsity to instance hardness. A second line of research is to examine also the performance of other metaheuristics on the LOP. However, preliminary results indicate that it will be very difficult to reach the performance of $ILS_{\mathcal{CK}}$.

Acknowledgments

The authors would wish to thank Prof. John Mitchell and Dr. Brian Borchers for making available the code of the simplex. This work was supported by the "Metaheuristics Network", a Research Training

[1] These instances will be made available at the address `http://www.intellektik.informatik.tu-darmstadt.de/~schiavin/lop`.

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. O. Becker. Das Helmstädtersche Reihenfolgeproblem – die Effizienz verschiedener Näherungsverfahren. In *Computer uses in the Social Science*, Wien, January 1967.
2. 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.
3. S. Chanas and P. Kobylanski. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6:191–205, 1996.
4. T. Christof and G. Reinelt. Low-dimensional linear ordering polytopes. Technical report, University of Heidelberg, Germany, 1997.
5. R. K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimisation*. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK, 2000.
6. M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, November-December 1984.
7. M. Grötschel, M. Jünger, and G. Reinelt. Optimal triangulation of large real world input–output matrices. *Statistische Hefte*, 25:261–295, 1984.
8. 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.
9. D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison Wesley, New York, 1993.
10. M. Laguna, R. Martí, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operation Research*, 26:1217–1230, 1999.
11. H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. Technical Report AIDA–00–06, FG Intellektik, FB Informatik, TU Darmstadt, November 2000. To appear in F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, Kluwer, 2002.
12. 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.
13. J. E. Mitchell. Generating linear ordering problems. `http://www.rpi.edu/~mitchj/generators/linord`, November 2002.
14. 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.
15. C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operational Research*, 86:473–490, 1999.
16. G. Reinelt. Library for linear ordering problem. `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/`, November 2002.
17. 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.
18. P. Stadler. Landscapes and their correlation functions. *J. of Math. Chemistry*, 20:1–45, 1996.
19. Standard Performance Evaluation Corporation. SPEC CPU95 and CPU2000 Benchmarks. `http://www.spec.org/`, November 2002.
20. E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.
21. S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proc. of the Sixth Congress on Genetics*, page 365, 1932.