Search Space Analisys for the Linear Ordering Problem

Tommaso Schiavinotto, 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-worls 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 indices $\{1, ..., n\}$ such that the value

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

is maximised. 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 maximised.

Both exact algorithms [6, 4, 10] 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 [8], 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 `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/LOLIB/`; for all LOLIB instances optimal solutions are known [7]. Because LOLIB instances are rather small, Mitchell and Borchers [10] 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 `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). Other instances were based on random generators [8], 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 [9] 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 previoulsy proposed metaheuristic approaches.

The paper is structured as follows. In the next section we analyse the behaviour of different local search algorithms on the two instance classes. Section **??** gives 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, $Insert : \Pi \times \{1, \ldots, n\}^2 \to \Pi$, is defined for $i \neq j$:

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

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. $Interchange : \Pi \times \{1, \ldots, n\}^2 \to \Pi$, for $i \neq j$:

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

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

$$(sort^* \circ reverse)^* \circ sort^*$$

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.

Tables 1 and 2 show the results obtained by applying the various local search 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 LS in the $\mathcal{N}_X$ neighbourhood returns poor quality solutions using comparable computation times to *insert*. The VND algorithms shows that only by concatenating

Table 1. Comparison of first-improvement algorithms based on different neighbourhoods on MBLB instances (B.I. indicates that, differently from the default, 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
$\mathcal{N}_I$	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

Table 2. Comparison of different LS on the 49 LOLIB instances; for an explanation of the table entries we refer to the caption to Table 1.

Local Search	Avg. Dev. (%)	# Opt.	time (s)	# Inst. Improved	% Improved Runs
Insert	0.1842	42	0.1802	–	–
$\mathcal{CK}$	0.2403	38	**0.0205**	–	–
Insert + $\mathcal{CK}$	**0.1819**	**44**	0.1881	**33**	**8.37**
$\mathcal{CK}$+ *Insert*	0.2360	40	0.0420	2	0.08

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 LS and that the improvement through VND is not important (besides the case of *interchange+insert*). Hence using a plain LS 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.

3 Structural analisys of the instances

The analysis that we are going to present has the goal to find out major differences on the instances that we used for experiments: LOLIB and MBLB.

LOLIB is made of 49 real world examples, there are 39 instances of size 44, 5 of size 50, 11 of size 56 and 3 of size 60. MBLB is randomly generated (from an uniform distribution) and then certain amount of zeroes are add; it is composed of 30 instances: 5 instances of size 100, 10 of size 150, 10 of size 200 and 5 of size 250.

Table 3. LOLIB 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	2nd Qu.	Max	Mean
44	Sparsity	0.00	21.53	41.27	55.06	80.63	39.36
	C. o V.	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
	C. o V.	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
	C. o V.	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
	C. o V.	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

Beside the size some other measures have been taken over the different instances, in order to describe the their structure.

The sparsity (percentage of elements that are equal to zero) seemed to be a very influent feature for the experiments presented in [10]. While the real world examples had a very variable sparsity (from 0% to 80%), MBLB where generated with fixed sparsity of 0%,10%,20%. For each instance it has been also computed two other measures that depend also on the sparsity. The first is the *Coefficient of Variance*, that 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 esteem of the variance that is indipendent from the size and the scale of the entries. . The other measure is the skewness, a coefficient of the data calculated on the third moment of the mean and normalized by the standard deviation. This coefficient indicates how much the entries distribution is asymmetric. In Tab. 3, 4 describe some information about this measures. For the two classes of instances data are represented in two different ways, because of the low variance of this values on the MBLB instances, and furthermore this class easily allows to group on the sparsity. As we see the latter measure determines the skewness and the CV for MBLB instances. We could expect such behaviour because the MBLB where generated by an uniform distribution.

As we can see from the gathered data the MBLB library is composed by similar instances, while the structure for the LOLIB instances is not well defined, as we expected as real world instances.

So far we limited the study to the instances' entries, in the next section we present a study about some features of the search space.

Table 4. Ranges of values for the coefficient of variance, and the skewness measure for MBLB, classified over sparsity and size.

Coefficient of variance

Size	Sparsity 0%	Sparsity 10%	Sparsity 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%	Sparsity 10%	Sparsity 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	-	-

4 Landscape analisys

Landscape analysis is an instrument used in order to study not trivial features of combinatorial problems. This type of analysis can be traced back to [14] for the study of biological evolutionary systems.

The idea is to "visualize" the search space as a landscape formed by all the solutions (in our case permutations), then for each one the quality of the solution (*fitness*) gives the altitude of the corresponding point. In our case, the quality of the solution corresponds to the objective function f, hence, the optimization process looks in that landscape for the highest point (best solution).

Formally, a landscape (for LOP) is described by a triple $\langle \Pi(n), f, d \rangle$, where Π is the set of all permutations of the sequence $(1, \ldots, n)$, f is the cost function and d is a distance that has to be defined. The distance d 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 local search we are using. In this work we focus on the *insert* local search, and the $\mathcal{CK}$ algorithm. For each one the distance is defined as the minimum number of applications of the basic local search operation to pass from a permutation to another. While for the $\mathcal{CK}$ algorithm it is not clear which this *basic operation* is, in the case of the *insert* local search it is clearly the *insert* operation. Unfortunately, at least at our knowledge, it seems that is not possible to obtain the number of minimum *insert* applications, in order to transform a permutation into another one, in an effiecient way. Because of this we used a surrogate distance that evaluates the difference of two permutation just considering the position of the elements in both. 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 $\frac{n(n-1)}{2}$. Hence, the maximum distance between two sequences $\frac{n(n-1)}{2}$, this value corresponds also to the *diameter* of the landscape.

One of the features of a fitness landscape that is often investigated is the ruggedness: a fitness landscape is said to be rugged if there is a low correlation between neighboring points. In order to calculate this correlation it has been suggested [12, 11, 13] 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 $r(s)$, that is 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 empirically calculated this measure on the instances we used, the neighborhood used is $\mathcal{N}_I$, the $\mathcal{CK}$, as we already stated, has an ill defined neighborhood that does not allow such a kind of analisys. For LOLIB instances l was in the interval of 75-84% of the problem size (n), the same measure for MBLB instances layed in the 93-98% of the problem size. This seems to indicate that, abstracting from the instance size, the former instances are harder for the *insert* move than the latter.

A this point it has been generated a large amount of local optima for all the instances (13000 for the LOLIB, 1000 for the MBLB).

A first analisys that we made over this data we generated is on the number of global optima that has been found among the l.o. generated: if we consider, for instance, the local optima generated by the *insert* LS we obtain that the percentage of distinct global optima over the distinct local optima generated is 0.46% for MBLB (min. 0%, max. 4.27%) and 14.10% for LOLIB (min 0.47%, max 85.12%). Furthermore, as Tab. 1 and 2 show, the l.o. found have a smaller deviation from the optimal value for the MBLB. This results confirm what was in the intention of Mitchell and Borchers at the moment of generating the MBLB instances: to have few global optima and a large amount of local optima with a close value of the objective function.

Another standard technique in analyzing the landscape is the study of the correlation between the quality of the local optima and the distance from the global optima. If suct a correlation exists and is positive, then an algorithms that aim to improve the solution quality obtains also the effect of moving towards the region of the search space where global optima lay. The correlation is a value between -1 and 1, in minimization problems a positive value shows a positive correlation, we computed such a value on the distance and the deviation from the optimal value (that we want to minimize).

A t-test stated that the value obtained by $\mathcal{CK}$ LS and *insert* LS are not significantly different both for LOLIB (average correlation: for *insert* 0.6189; for $\mathcal{CK}$ 0.6274) and for MBLB (average correlation: for *insert* 0.7867; for $\mathcal{CK}$ 0.7858). As so far happened the variance of this measure is higher on LOLIB instances than on MBLB, because the structures of the instances are extremely different in the former class, for the *insert* LS, the correlation values range between -0.11 and 1 on LOLIB instances, and between 0.61 and 0.94 on MBLB. MBLB instances show, overall, a better correlation between distance and solution quality. Plots in Fig. 1 represent graphically such correlation over two instances for LOLIB with the *insert* LS.

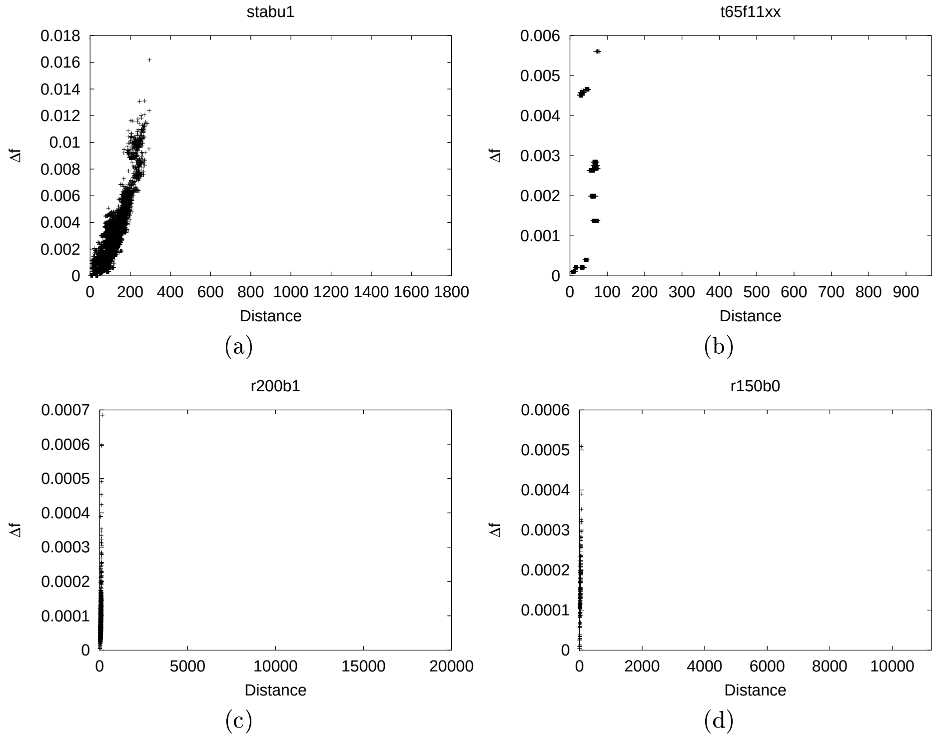


Fig. 1. Two plots of the deviation from the optimal value versus the distance of the generated local optima. On the left an instance with an high correlation value (0.8757), on the right one with a low correlation value (-0.1056).

The search space study we did so far suggests that the LOLIB instances represent an harder class of problem that MBLB. We make some consideration about this when we will draw our conclusions. In the next chapter we show the results we obtained with two iterated local search metaheuristic: one based on $\mathcal{CK}$ LS and the other on *insert* LS.

5 Iterated Local Search

ILS is a simple metaheuristic that showed to achieve good results on several combinatorial problem []. It tries to enhance the quality of the solution obtained by local search through iterating over the local search process. This is done by: generating a new solution starting from a local optima; moving over a neighborhood that is different from the one used in the local search (*perturbation*); applying newly the local search process. In both the implementations we present here, the perturbation is made using several *interchange* operations. Actually, the only difference between the two implementations are similar, is that one is based on $\mathcal{CK}$ LS ($\text{ILS}_{\mathcal{CK}}$) and the other on the *insert* LS (ILS_I). The parameter tuning has

	Size	Min	1st Qu.	Median	2nd 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

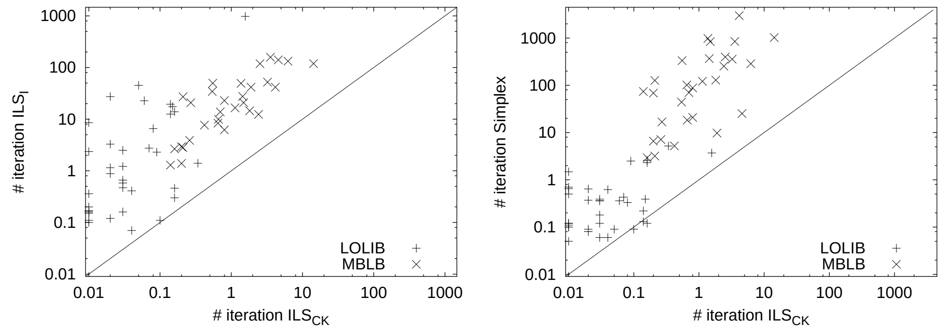


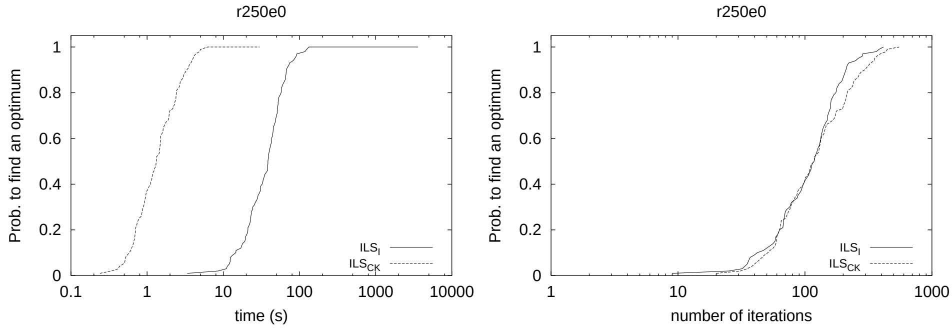
Fig. 2. Pairwise comparison of $\mathrm{ILS}_{\mathcal{CK}}$ with ILS_I and the Simplex algorithm.

been done on the MBLB instances, for $\mathrm{ILS}_{\mathcal{CK}}$ the same settings were good also for LOLIB instances, while the ILS_I showed a lack of tuning on the latter class of instances.

For each algorithm 100 runs has been made over the instances. On LOLIB instances $\mathrm{ILS}_{\mathcal{CK}}$ was able to find always a global optimum in less than 0.34s, beside one instance (`be75np`) where the maximum time for finding a global optimum is of 1.57s). On the MBLB instances instead it takes 14.27s for the "r200e1", and less than the half for all the others. For the former class of problems this algorithms seems to be at least comparable to the results presented in [2] where a "scatter search" algorithm were used, and in [8] where an "elite tabu search" were used. Indeed, it is not possible to do straight comparisons with these previous results, both because of different machine used, both because the meaning of the published results is not clear.

ILS_I resulted worse than the other ILS implementation, in fact the timings for finding always the optima all over the 100 runs range between 1.29s and 158.33s for the MBLB, and between 0.02s and 45.19s with a peak of 977s for the `be75np` instance (such a behaviour can be in part due to parameter under-tuning).

As control, we used a simplex algorithm that Mitchell and Borchers implemented, and used in their paper [10] as comparison with their algorithm. Fig. 2 illustrates a pairwise comparison between the algorithms. A point in the plot is given by comparing the maximal time for finding a global optima all over the 100 runs. It is important to notice, that the comparison with the simplex is somehow



(a) (b)

Fig. 3. Runtime distribution plots on a MBLB instance: over time (a); over number of iterations (b).

inaccurate, because the simplex is an exact algorithm, while the ILS does not assure that the optima will be found. ILS$_{\mathcal{CK}}$ results to be the best one between the three algorithms.

Runtime distribution (RTD) plots illustrate the probability of finding a global optima over the running time. This kind of plots on MBLB instances (for instance Fig. 3(a)) showed an interesting behaviour of the two implementations: the shape of the curve seems to be similar. This suggested of doing the same plots but over the number of iterations made by the two ILS (Fig. 3(b)). What we obtained is that if we consider the number of iterations the two algorithms behave similarly, and some time ILS$_I$ needs less iterations than the other. In the case shown if Fig. 3 ILS$_I$ needs 132.99s against 6.25s of ILS$_{\mathcal{CK}}$, but from the other point of view ILS$_I$ needs 416 iterations, less than ILS$_{\mathcal{CK}}$ that needs 552 iterations. This result show (at least on MBLB instances) that the difference between the two local searches is mostly given by the speed, and not by the quality.

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), bootstraping 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, so a lower value than the original instances. We choosed an instance (`be75np`) that resulted to be hard in the original problems. The simplex was not finished after 8500 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.

6 Conclusions

In this paper we proposed a study on the search space of the Linear Ordering Problem. We particularly focused on the LOLIB and MBLB classes of instances.

The logic lead Mitchell and Borchers to generate the instances of MBLB, where the local optima have objective function values close to those of the global optima and where the number of the latter is low. However our study suggested that LOLIB class of problems can be harder, this has been also confirmed by some preliminary experiment that resulted surprisigly hard.

We also presented an implementation of iterated local search that use the $\mathcal{CK}$ heuristic (described in [3]), this algorithm performances are at least comparable with the existing metaheuristic implemented so far for the the LOP. This local search resulted to be faster than the one based on *insert*, the latter is a more standard LS with a well defined neighborhood. The speed is the major advantage brought by the new LS, because it allows to visit in the same time a larger amount of solutions, while the solution quality does not improve significantly.

We can state that LOP is not a trivial problem, even if real world problem are not large enough to be challenging we can use them to generate instances that will be hard to resolve.

A future work on the search space analisys for this problem should try to individuate the features that most influences the hardness of the problem more directly. For instance, it is our hypotesis that the sparsity has a very important role in regulating the hardness of the problem, because of the influence that it will have on the size and number of plateux in the search space.

7 Acknowledgements

The authors would wish to thank Prof. John Mitchell and Dr. Brian Borchers for the code of the simplex they made available to us.

References

1. O. Becker. Das Helmstädtersche Reihenfolgeproblem – die Effizienz verschiedener Näherungsverfahren. In Bericht einer Working Conference, editor, *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 M. Dorigo D. Corne and F. Glover, editors, *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 XXX, Department of Mathematics and Computer Science, 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. 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.
9. 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.
10. 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.
11. Peter F. 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.
12. Peter F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996.
13. E. D. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.
14. S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth Congress on Genetics*, page 365, 1932.