Perturbation: An Efficient Technique for the Solution of Very Large Instances of the Euclidean TSP[*]

B. Codenotti[†], G. Manzini[‡], L. Margara[§] and G. Resta[¶]

TR-93-035

July 1993

Abstract

In this paper we introduce a technique for building efficient iterated local search procedures. This technique, that we call *perturbation*, uses global information on TSP instances to speed-up and improve the quality of the tours found by heuristic methods. The main idea is to *escape* from local optima by introducing perturbations in the problem instance rather than in the solution. The performance of our techniques has been tested and compared with known methods. To this end, we performed a number of experiments both on test instances, for which the optimal tour length is known, and on uniformly distributed instances, for which the comparison is done with the Held-Karp lower bound. The experimental results, done on up to 100,000 cities, show that our techniques outperform the known methods for iterating local search for very large instances.

[*]This work has been partially supported by the CNR through the "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo. Sottoprogetto 2". G. Manzini and G. Resta have been partially supported by an INDAM postdoc fellowship.

[†]ICSI, 1947 Center Street, Suite 600, Berkeley, CA 94704-1105

[‡]IEI-CNR, Via S.Maria, 46 56100-Pisa (Italy)

[§]Dipartimento di Informatica, Università degli studi di Pisa (Italy)

[¶]IEI-CNR, Via S.Maria, 46 56100-Pisa (Italy)

1 Introduction

Given N cities $i = 1, \ldots, N$, separated by distances d_{ij} the (euclidean) Traveling Salesman Problem — TSP from now — consists of finding the shortest closed path visiting each city exactly once. The solution of very large instances of the TSP has challenged several authors over the last few years [1, 2, 5, 7, 10]. The results have been quite satisfactory. In fact, by using, e.g., the Lin-Kernighan method — LK from now — it is possible to face TSP instances with thousands of cities and obtain, within a reasonable time, tours which are very close to the optimal one. On the other hand we are still far from solving the general problem of evaluating the performance of local search heuristics and capturing the mathematical properties of the correspondent local optima. For some preliminary results, see [8].

The main contributions in the field of experimental solution of large TSP instances come from David Johnson and several coauthors [3, 7]. To the best of our knowledge, there are no other significant results for instances with more than 1,000 cities. This paper provides a new framework for implementing iterated local search so that it becomes feasible to find almost optimal solutions to TSP instances with more than 100,000 cities.

The problem of gathering some global information on an instance of the TSP seems to be central; in fact all the methods which avoid this are characterized by either a significant loss of precision or a running time penalty.

In this paper, we continue the work started in [4] and develop an adaptive framework based on a global parameter, the *problem sensitivity*, whose evaluation suggests a number of strategies to improve over existing methods. More precisely, we implement iterated local search by transforming the instance at hand into a different one. There have been several attempts to formalize some intuitions on the structure of the local optima found by local search procedures (see e.g. [8]). We use the notion of problem sensitivity as the theoretical background for devising a perturbation strategy to be used in the iteration of local search procedures.

Sensitivity has been widely recognized as one important parameter in the analysis of computational problems. It consists of measuring how the output of the problem changes upon slight changes in the input. For the TSP this property could be interpreted as the relationship between the optimal tour of a given problem instance and the optimal tour of a slightly different one, e.g. with a small modification in the matrix of distances. It could also be used to analyze the modifications in the local optima. This viewpoint leads to the idea of moving from a given instance to another one as shown in Fig. 1.

More precisely, given a local optimum s for a problem P, we consider a "perturbed" problem P' and a one-to-one correspondence f between the cities in P and P'. We then construct a solution s' for P' as $s' = f(s)$. s' needs not to be a local optimum for P' so one can apply local search — with s' as starting point — to get a local optimum s'' for P'. This can then be mapped onto a solution $t = f^{-1}(s'')$ for P which again is

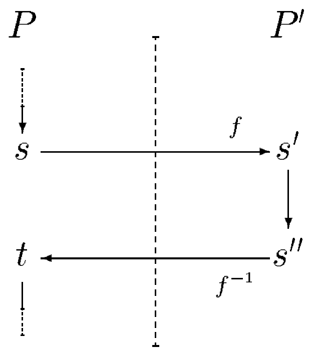

Figure 1: We start with a local optimum s for the original problem P. We apply a transformation f to s obtaining a solution s' for a perturbed problem P'. In general, s' it is not a local optimum for P'; starting from s' we obtain a local optimum s''. Inverting the trasformation f we convert s'' into a solution t for P, which, again, is not necessarily a local optimum, so that local search can be re-applied.

not — in general — a local optimum for P. Thus local search can start again.

The key feature of this approach is the fact that we perturb problem instances rather than solutions so that we can map local optima onto good solutions which can be eventually improved by local search, since they are not necessarily local optima. This approach can be a first step towards the development of novel intuitions about how to speed-up local search and how to *escape* from local optima.

The rest of this paper is organized as follows. In Sect. 2 we present the perturbation method. In Sect. 3 we discuss some implementation issues. In Sect. 4 we report on the experimental results, and Sect. 5 contains some concluding remarks.

2 The Method of Perturbation

LK is the most accurate local search heuristic for the TSP and finds tours whose lengths are, on the average, 2.1% off the optimum. The only known technique which allows one to find tours which are significantly shorter than those found by single applications of local search is *Iterated Local Search* (ILS). ILS works as follows:

ILS.
```
1. Find an initial solution $s$ by using local search.
2. Do the following for a given number $M$ of iterations.
   2.1.Perturb $s$ obtaining a new solution $t$.
   2.2.Run local search on $t$ obtaining $u$.
   2.3.If $length(u) < length(s)$, set $s = u$.
4. Return $s$.
```

The effectiveness of ILS mainly depends on two factors: the perturbation strategy used in Step 2.1 and the local search procedure used in Step 2.2. In practice, ILS finds solutions much shorter than those found by *repeated local search*, which consists of running the local search procedure for a certain number of times, starting from independently chosen initial tours.

ILS has been introduced by Martin, Otto, and Felten [11]. They initially tested this technique on the euclidean TSP by using 3-Opt local search procedure in Step 2.2. To perturb the solution s - Step 2.1 - they remove 4 arcs from s and replace them in order to obtain a non-sequential move. More precisely, they first replace two arcs belonging to s by the two arcs which disconnect the tour. Then they repeat the same operation in order to reconnect the tour (see [11]). This kind of perturbation move is called *Double Bridge*. (From now on, DB-ILS will denote the ILS procedure using the Double Bridge move.) They also introduced (see step 2.3 above) the possibility of accepting solutions of increasing cost with a certain probability, using a technique which is similar to simulated annealing. Johnson [7] has investigated the improvement of ILS when 3-Opt local search is replaced by the more powerful LK heuristic in Step 2.2.

Local search heuristics take a solution s as input and yield a local optimum w which is shorter than s. Experimental results confirm the intuitive fact that the quality of w strongly depends on the quality of s. As an example, local search applied to an initial solution obtained by a direct method, e.g., multiple fragment heuristic [2], finds solutions which are better than those obtained by applying the same local search procedure to random initial tours. ILS takes advantage of this experimental evidence by applying local search to solutions which are obtained by perturbing a previously found local optimum.

Intuitively, one has to look for a perturbation strategy which neither produces a tour t which is very long, nor keeps substantially the same structure as s. In fact, in the former case, local search applied to t is not likely to find a solution shorter than s, and, in the latter case, local search stops soon and does not lead to a significant improvement.

Informally, a good perturbation strategy should produce a solution t such that: (i) the length of t is close to the length of s, and (ii) t is quite different from any local optimum (in particular from s). If t satisfies properties (i) and (ii), then it is conceivable that local search, applied to t, yields a local optimum u better than s.

Here we present a new perturbation strategy which produces perturbed solutions satisfying (i) and (ii). Our main idea consists of working on *perturbed problems* instead of perturbed solutions. More precisely, we perform Step 2.1 as follows.

Step 2.1.
```
a. Find a new set $P'$ of cities by applying a small random
   perturbation to $P$.
b. Starting from the solution $s$ for $P$, find the
```

```
   correspondent solution $s'$ for $P'$
c. Find a new solution $s''$ by applying a local search
   procedure to $P'$ starting from $s'$.
d. Starting from the solution $s''$ for $P'$, find the
   correspondent solution $t$ for $P$.
```

Let π be a permutation of $1, \ldots, n$, $P = \{p_1 \ldots, p_n\}$ be a set of cities and $s = p_{\pi(1)}, \ldots\ p_{\pi(n)}$ be a solution for P.

In Step a we find a perturbed problem P'. This can be done, for example, moving $p_1 \ldots, p_n$ by ϵ or removing a certain number k of cities from P. In Step b we find a solution s' for P' starting from s. As an example, if we perturb P by moving the cities by ϵ, s' is equal to s.

Since P' is different from P, s' is not a local optimum for P'. In Step c we apply a local search procedure to s' obtaining a local optimum s'' for P'. In Step d, starting from s'', we find the correspondent solution t for P, where t is not necessarily a local optimum for P. Experimental results show that the solution t is quite different from s although its cost is close to the cost of s. This perturbation strategy allows the local search procedure to proceed even if a local optimum for P has already been found.

Local search runs temporarily on a perturbed problem P'. When no further improvement can be done on P', local search turns again to the original problem P. The intuition behind this approach is visualized in Fig. 1.

We propose two different ways to introduce perturbations, which we call *ϵ-move* and *k-remove*.

ϵ-move. Given a euclidean TSP $P = \{p_1, \ldots, p_n\}$, this strategy produces a new problem $P' = \{p'_1, \ldots, p'_n\}$ such that $dist(p_i, p'_i) = \epsilon_i$, $\forall i$, $1 \leq i \leq n$, where $dist(a, b)$ stands for the euclidean distance between the points a and b. We choose the value of ϵ_i equal to $1/5$ of the sum of the distances of p_i from its neightbors in the actual tour. This allows us to perturb the problem according to the quality of the solution found so far. Each city moves towards a randomly chosen direction. One can readily verify that any solution s for P is also a feasible solution for P', and viceversa.

k-remove. This strategy produces a new graph P' simply by removing from P a certain number $k << n$ of cities. Any solution s for P can be translated into a solution s' for P' by disregarding the removed cities. The inverse operation is slightly more complicated. In fact we have to take into account the requirements expressed by properties (i) and (ii). We adopt the following strategy. Each city $p_{\pi(i)}$ is inserted between two other cities $p_{\pi(j)}$, $p_{\pi(j+1)}$ belonging to s'', chosen at random among the first m neighbors of $p_{\pi(i)}$.

Once a perturbed solution s' has been found (Step 2.1), ILS runs local search on s' in order to obtain a local optimum s'' for the perturbed problem, and then a new local optimum u for the original problem. At this point, ILS accepts or rejects u depending on its length. If u is shorter than s, then ILS sets $s = u$. We perform this step by allowing a low probability acceptance for tours u longer than s. Let $\delta = (length(u) - length(s))/length(u)$. We accept solution u with probability $p =$

$c_1 \cdot e^{-c_2 \delta}$, where c_1 and c_2 are suitable constants. Experimental results show that this mechanism enables local search to find shorter tours than those found by using the criterium described in Step 2.3.

3 Implementation Issues

We have implemented and extensively tested both the ϵ-move and the k-remove perturbation methods. In this section we give some details on the implementation of ILS, and we describe some techniques used to speed-up the code. In the next section we discuss the experimental results.

The most successful local search procedures for the TSP are the 2-Opt, 3-Opt, and LK algorithms (see [2][9] for a complete description of these methods). Unfortunately, it does not exist any theoretical estimate of the quality of the solutions found by these procedures. However, they have been extensively tested for many years and their performance *in practice* is well known. As an example, for random Euclidean instances with 10,000 cities, our implementation of 2-Opt finds solutions that are on the average 5.17% off the Held-Karp lower bound[1], and for the 3-Opt and LK procedures this percentage reduces to 3.16% and 1.96%.

The naive implementations of 2-Opt, 3-Opt and LK algorithms take $O(N^2)$, $O(N^3)$, and $O(N^5)$ time, respectively, where N is the number of cities. However, by using appropriate data structures and by taking advantage of certain geometric properties of the tours, it is possible to reduce substantially the running time of these algorithms [2]. Further significant reductions of the time complexity are possible by implementing "approximate" algorithms that find slightly longer tours but are much faster than the original versions. These approximate algorithms are usually characterized by the utilization of "clever" data-structures and programming "tricks" that are *necessary* in order to apply, within reasonable time limits, local search procedures to TSP instances with 10,000 cities and more. This is particularly true for ILS algorithms where local search is repeated many times.

In order to substantially reduce the running time of local search procedures we use the *don't-look bit* technique described in [2, Sec. 4]. Our LK algorithm always uses this technique, while 2-Opt and 3-Opt algorithms use don't-look bits only when applied to perturbed problems.

In our first experiments we have implemented the ILS algorithms based on the ϵ-move strategy, and have performed local search at steps 2.1.c and 2.2 using 2-Opt, 3-Opt, and LK. One of the basic operation of all local search procedures consists of finding all cities within a certain radius of a given city. In order to perform efficiently this near neighbor search, our algorithms execute a preprocessing stage in which the following data structures are created:

[1]Held and Karp [6] have proposed an iterative technique based on minimum spanning trees which produces sharp lower bounds on the optimal tour length.

1. a bidimensional array `near[][]` such that `near[i][·]` is the list of the 20 cities closer to city `i`, sorted by increasing distance;

2. a bidimensional array `dist[][]` such that `dist[i][j]` is the distance between cities `i` and `near[i][j]`.

Unfortunately, this data structure can not be used in step 2.1.c of ILS algorithm with ϵ-move strategy. In fact, during step 2.1.c local search is performed on a *perturbed* problem P' that changes at each iteration. Therefore, the distances between cities are different from the distances contained in `dist[][]` (however, note that the distances change only slightly).

For each perturbed problem P' it would be necessary to compute a new pair of arrays `aux_near[][]` and `aux_dist[][]`. In order to speed up the algorithm we have used the data contained in `near[][]` also for the perturbed problems P'. Clearly, for P' the cities contained in `near[i][]` are simply a set of cities close to city `i`, but in general they are *not* the *more close* cities to city `i`. This implies that in general the solution s'' found at step 2.1.c it is not a local optimum. However, since s'' must be transformed into a solution t for the problem P, the length of s'' is not critical and also a non-optimal solution can be accepted.

The distances between cities `i` and `near[i][j]` (for the problem P') are stored in `aux_dist[i][j]`. For these values we utilize a "lazy evaluation" scheme: distances are computed only when required by the algorithm. This guarantees that each distance is computed only once, and that useless distances are not computed.

Moreover, in order to obtain an efficient implementation of ILS with 3-Opt and LK, we have used a modified ILS algorithm in which the local search at step 2.1.c is performed by using the 2-Opt procedure instead of the 3-Opt or the LK algorithm. In fact, as we have already pointed out, the length of the solution s'' found at step 2.1.c is not critical.

The implementation of the k-remove strategy is more straightforward. The data contained in the arrays `near[][]` and `dist[][]` can be utilized also for the perturbed problem P'; we only need to handle properly the cities that have been removed. We have tested the algorithm using 2-Opt, 3-Opt, and LK procedures for the local search at steps 2.1.c and 2.2.

The experimental results reported in the following section have been obtained by removing k cities from P, where $k = \lceil \text{number of cities}/200 \rceil$. The parameter 200 has been chosen on the basis of several tests performed with different values of k.

4 Experimental Results

We have extensively tested our pertubation methods — ϵ-move and k-remove— both on random uniformly-distributed and known TSP instances. We have compared the quality of the solutions provided by our methods with those found by the DB-ILS procedure which is the best known method for iterating local search. Each algorithm (ϵ-move, k-remove, and DB-ILS) has been tested with 2-Opt, 3-Opt and LK local search procedures. All tests have been performed on a Silicon Graphics Iris Indigo R4000 with 32 megabytes of main memory.

Table 1 gives the running time and the percentage excess over the Held-Karp lower bound for a single application of the 2-Opt, 3-Opt and Lin-Kernighan algorithms. For all instances the initial solution has been obtained by using the Multiple Fragment (MF) tour construction heuristic. Tests are performed on random euclidean instances with up to 100,000 cities.

Table 2, 3,and 4 report on the percentage excess over the optimal tour length (extimated by using the Held-Karp lower bound) of the solutions found by ILS procedures DB-ILS, ϵ-move, and k-remove, based on 2-Opt, 3-Opt, and LK, respectively. Tests are performed on random euclidean instances. Table 5 reports on the running time of a single iteration of the above described techniques.

The running times we report do not include the time spent for the preprocessing, which is shared by all the methods we test. During the preprocessing we find the 20 nearest neightbors of each city. This takes a time that is roughly linear in the number of the cities, ranging from 0.3 seconds for 1,000 cities to 56 seconds for 100,000 cities.

Note that the running time of the first iteration (Table 1) is much bigger than the average running time of the subsequent iterations (Table 5). This is due to the fact that the first iteration takes as input a very long solution (the one produced by MF heuristic whose length is about 16% over the Held-Karp lower bound) and yields a quite good solution. While all the other iterations only slightly improve the current solution by producing small changes. This makes iterated local search — which already is more accurate — also much faster than repeated local search.

Table 6 gives the results — for DB-ILS, ϵ-move, k-remove — on some instances (taken from TSPLIB [12]) for which the optimal tour length is known. The local search procedure used is LK.

Fig. 2 gives the performance of these algorithms as a function of the actual running time for a 100,000 city instance. Also in this case we only report on the experimental results obtained by using LK heuristic.

A general result is that the effectiveness of ILS decreases as the number of cities increases. The main issue to this regard is to compare the different strategies. The one suffering more from the above inconvenience is DB-ILS. In fact, it does not give any improvement after 100 iterations with the 2-Opt strategy, for more than 1,000 cities (table 2), after 100 iterations with the 3-Opt strategy (table 3) for more than 5,000 cities and after 500 iterations with LK (table 4) for 10,000 cities. The performance of our methods is much better. In particular, ϵ-move is able to improve the solution even after 500 iterations, and is the only method that allows one to improve local search performed by 2-Opt. This is a quite surprising result because it makes feasible to run several times the 2-Opt strategy and achieve tours less than 3% off the optimum.

Although our methods are slightly slower than DB-ILS, note that 100 iterations

Cities	Running time in seconds				Percentage excess			
	MF	2-Opt	3-Opt	LK	MF	2-Opt	3-Opt	LK
1,000	0.10	0.10	0.44	2.83	17.41	6.46	3.26	2.16
5,000	1.09	1.05	6.42	26.38	16.37	5.82	3.36	2.14
10,000	2.65	4.75	23.55	68.29	15.61	5.17	3.16	1.96
50,000	34.61	45.79	486.89	1364.36	15.02	5.59	3.34	2.32
100,000	120.15	203.31	2139.67	5566.53	14.86	5.31	3.21	2.16

Table 1: Running time in seconds and percentage excess over the Held-Karp lower bound for the 2-Opt, 3-Opt and Lin-Kernighan algorithms. For all instances the initial solution has been obtained by using the Multiple Fragment (MF) tour construction heuristic.

of our methods almost always yield better tours than $1,000$ iterations of DB-ILS, for more than $5,000$ cities, and thus they are superior also for what concerns the overall running time of the ILS procedure.

For instances with 100,000 cities the gap between the performance of our methods and DB-ILS becomes more evident. In fact, Fig. 2 shows that DB-ILS, after ten hours, improves the solution provided by LK only by 0.02% (note that the last improvement is obtained after about 1.9 hours). Our perturbation techniques allow to progressively and continuously improve the solution at hand (improvements are obtained also after 9.8 hours).

A direct consequence of this fact, is that our perturbation techniques can take advantage of larger amounts of time or, equivalently, of faster machines and code optimization. DB-ILS seems not to enjoy this property.

5 Conclusions and Further Work

In this paper we have proposed some alternatives to existing techniques for iterating local search procedures for the TSP. The behaviour of our approach has been experimentally evaluated both on random instances and on test problems. To our knowledge, the results of this paper are the best in the field of ILS.

Further work to be done includes the implementation of the different methods on parallel machines. In addition, we are currently exploring the features of different perturbation techniques. We hope that the idea of working on perturbed instances could lead to new clues on the structure of local optima.

Acknowledgement. We would like to thank David Johnson for some suggestions and for providing some benchmarks.

Cities	**2-Opt**	Percentage excess after 100, 500, and 1000 iterations								
		DB-ILS			k-remove			ϵ-move		
		100	500	1000	100	500	1000	100	500	1000
1000	6.46	4.96	4.40	4.24	5.19	4.95	4.82	3.62	2.93	2.68
5,000	5.82	5.40	5.40	5.40	5.52	5.52	5.52	3.62	3.14	2.94
10,000	5.17	4.93	4.93	4.93	5.08	5.08	5.08	3.20	2.89	2.89
50,000	5.59	5.39	5.39	-	5.59	5.59	-	3.66	3.04	-

Table 2: Average percentage excess over the Held-Karp lower bound for iterated local search methods after 100, 500 and 1,000 iterations, respectively. Column **2-Opt** gives the average percentage excess over the Held-Karp lower bound for the initial solution found by using 2-Opt local search.

Cities	**3-Opt**	Percentage excess after 100, 500, and 1000 iterations								
		DB-ILS			k-remove			ϵ-move		
		100	500	1000	100	500	1000	100	500	1000
1000	3.26	2.40	1.90	1.80	2.53	2.11	1.89	2.24	1.88	1.78
5,000	3.36	3.14	2.85	2.81	2.81	2.54	2.41	2.35	2.03	1.93
10,000	3.16	2.94	2.94	2.94	2.70	2.44	2.24	2.32	2.00	1.78
50,000	3.34	3.34	3.34	-	3.05	2.80	-	2.42	2.10	-

Table 3: Average percentage excess over the Held-Karp lower bound for iterated local search methods after 100, 500 and 1,000 iterations, respectively. Column **3-Opt** gives the average percentage excess over the Held-Karp lower bound for the initial solution found by using 3-Opt local search.

Cities	**LK**	Percentage excess after 100, 500, and 1000 iterations								
		DB-ILS			k-remove			ϵ-move		
		100	500	1000	100	500	1000	100	500	1000
1000	2.16	1.42	1.27	1.25	1.35	1.08	1.02	1.34	1.08	1.02
5,000	2.14	1.86	1.67	1.59	1.45	1.21	1.15	1.43	1.19	1.13
10,000	1.96	1.89	1.82	1.82	1.51	1.18	1.18	1.40	1.24	1.15
50,000	2.32	2.17	2.11	-	1.59	1.38	-	1.46	-	-

Table 4: Average percentage excess over the Held-Karp lower bound for iterated local search methods after 100, 500 and 1,000 iterations, respectively. Column **LK** gives the average percentage excess over the Held-Karp lower bound for the initial solution found by using Lin-Kernighan local search.

Cities	Running time per iteration								
	DB-ILS			k-remove			ϵ-move		
	2-Opt	3-Opt	LK	2-Opt	3-Opt	LK	2-Opt	3-Opt	LK
1000	0.07	0.18	0.52	0.03	0.10	0.58	0.06	0.12	1.05
5,000	0.91	1.94	3.48	0.28	1.05	4.75	0.35	1.40	12.97
10,000	2.63	5.61	8.05	0.74	3.49	10.38	0.87	4.91	40.75
50,000	41.20	86.79	77.87	10.13	68.46	249.92	7.24	104.56	771.59

Table 5: Average running time (in seconds) for a single iteration of different iterative methods.

Cities	**LK**	Percentage excess after 100, 500, and 1000 iterations								
		DB-ILS			k-remove			ϵ-move		
		100	500	1000	100	500	1000	100	500	1000
ATT 532	1.82	0.16	0.13	0.13	0.44	0.26	0.26	1.30	1.30	1.30
PR 1173	1.60	0.72	0.43	0.31	0.52	0.26	0.22	0.85	0.74	0.51
U 1432	2.29	0.78	0.62	0.62	1.12	0.30	0.27	0.55	0.27	0.25
PR 2392	1.32	0.72	0.51	0.51	0.83	0.55	0.46	0.78	0.74	0.74
PCB 3038*	2.33	1.82	1.82	1.52	1.51	1.10	1.01	1.47	1.26	1.11

Table 6: Percentage excess over the optimal tour length for known problems after 100, 500 and 1,000 iterations, respectively. *For the problem PCB 3038 the percentages refer to the lower bound 136,522 computed using the Held-Karp algorithm.

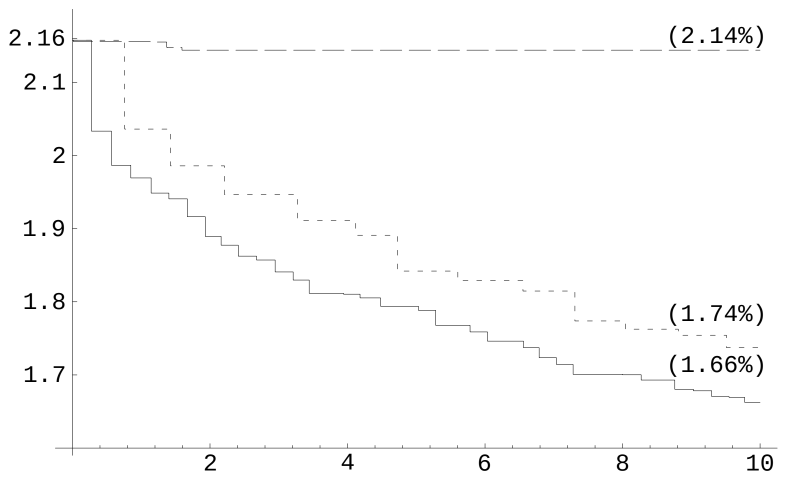

Figure 2: Average percentage excess over the Held-Karp lower bound versus running time, in hours, for a 100,000 cities instance. Dashed, dotted, and solid lines represent the behaviour of DB-ILS, ϵ-move, and k-remove, respectively. The percentages obtained after ten hours are displayed above each line.

References

[1] J. L. Bentley. Experiments on traveling salesman heuristics. *Proc. 1st Symp. on Discrete Algorithms*, 91–99, 1990.

[2] J. L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA. J. Comput.*, (4):387–411, 1992.

[3] J. L. Bentley, D. S. Johnson, L. A. McGeoch, E. E. Rothberg. Near-optimal solutions to very large traveling salesman problems. In preparation.

[4] B. Codenotti, G. Manzini, L. Margara, and G.Resta. Global strategies for augmenting the efficiency of TSP heuristics *Proc. of the 3rd Workshop on Algorithms and Data Structures,* August 1993.

[5] B. L. Golden, L. D. Doyle, W. Stewart JR. Approximate traveling salesman algorithm. *Oper. Res.*, (28):694–711, 1980.

[6] M. Held and R. Karp. The traveling salesman problem and minimum spanning trees. *Oper. Res.* 18:1138–1162, 1970.

[7] D. S. Johnson. Local optimization and the traveling salesman problem. *Proc. 17th Colloq. on Automata, Languages, and Programming, Lecture Notes in Computer Science 443.* , 446–461, 1990.

[8] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search ? *J. Comput. System Sci.* 37(1):79-100, 1988.

[9] S. Lin and W. Kernighan. An effective heuristic algorithm for traveling salesman problem. *Oper. Res.*, (21):493–515, 1973.

[10] E. Lawler, J. Lenstra, A. Rinnoy Kan, and D. Shmoys. *The traveling salesman problem.* John Wiley and Sons, 1985.

[11] O. Martin, S. W. Otto, and W. Felten. Large-step markov chains for the TSP incorporating local search heuristics. *Oper. Res. Lett.*, (11):219–224, 1992.

[12] G. Reinelt. TSPLIB — A traveling salesman problem library. *ORSA. J. Comput.*, (3):376–384, 1991.