Analyzing the Run-time Behaviour of Iterated Local Search for the TSP

Thomas Stützle[1] and Holger H. Hoos[1]

[1]Université Libre de Bruxelles, IRIDIA
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium.
E-mail: tstutzle@ulb.ac.be

[2]University of British Columbia, Computer Science Department
2366 Main Mall, Vancouver, BC, V6T 1Z4 Canada.
E-mail: hoos@cs.ubc.ca

1 Introduction

Metaheuristics strongly involve random decisions during the search process. Such random decisions are, for example, due to random initial solutions, randomized tie breaking criteria, randomized sampling of neighborhoods, probabilistic acceptance criteria and many more. Because of their nondeterministic nature, the run-time required by such an algorithm to achieve a specific goal — like finding an optimal solution to a given problem instance — is a random variable. Obviously, knowledge on the distribution of this random variable can give valuable information for the analysis and the characterization of an algorithm's behavior, provide a basis for the comparison of algorithms, and give hints on possible improvements of an algorithm's performance [8]. To obtain empirical knowledge on the run-time distribution (RTD) of an algorithm when applied to a specific problem instance, one may estimate the RTD from data collected over several runs of the algorithm and possibly approximate the empirically observed RTD by a distribution function known from probability theory. If similar behavior is observed on all tested instances from a particular problem class, for example, on Euclidean TSP instances, the observed type of RTDs characterizes the run-time behavior on this problem class. The RTDs may also give an indication under which conditions an algorithm may be improved. As we will later explain, the exponential distribution plays a crucial role for judging an algorithm's effectiveness.

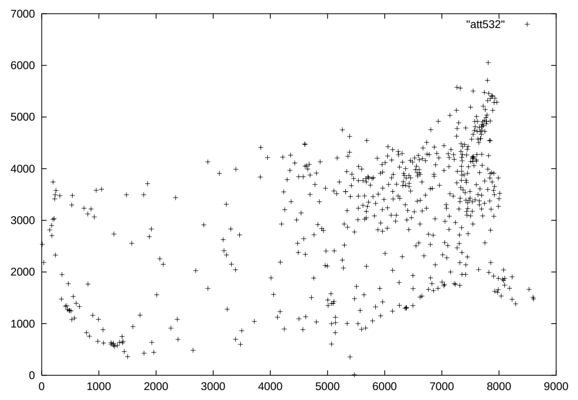
In this work we investigate the run-time behavior of local search algorithms for optimization problems, in particular, of Iterated Local Search (ILS) algorithms for the Traveling Salesman Problem (TSP). Iterated local search (ILS) [2, 17, 11, 24] is a very simple and powerful metaheuristic which has proved to be among the best performing approximation algorithms for the well known Traveling Salesman Problem (TSP) [16, 11]. ILS is based on the observation that iterative improvement local search is easily trapped in local minima. Instead of restarting the local search from a new, randomly generated solution, a better idea may be to modify the current solution, moving it to a point beyond the neighborhood searched by the local search algorithm. ILS then continues the local search from the so perturbed solution. An acceptance criterion determines to which local optimum the perturbation step is applied. In fact, in almost all ILS applications the solution modifications are always applied to the best solution found since the start of the algorithm [16, 20].

ILS algorithms make heavily use of random decisions and, in particular, the solution perturbations are randomized. Therefore, as argued before, measuring RTDs is potentially very helpful for analyzing ILS' performance. To empirically measure the RTDs we run a given algorithm many times on the same problem instance and collect some elementary data; in each run it suffices to report the solution quality whenever a new best solution is found, the computation time needed to obtain it, and possibly some other statistic data for further analysis. Then, a posteriori the empirical distributions for different bounds on the required solution quality can be easily estimated. Then, a posteriori the empirical distribution run-time distribution $G_c(t)$ to reach a solution quality bound c can be easily computed as $\widehat{G_c}(t) = | \{j \mid rt(j) \leq t \ \wedge \ f(s_j) \leq c\} \mid /k$. For some problems, optimal solutions or tight bounds on the optimal solution value may be known. In such a case it might be preferable to fix the solution quality bound c relative to the optimal solution value, that is, to require the algorithm to get within a certain percentage of the optimal solution value. The analysis of ILS run-time behavior reported in this work suggests improvements of the standard ILS algorithm; we exemplify the improved performance by giving some experimental results on known benchmark instances.

The paper is structured as follows. First, in Section 2 we introduce the TSP. In the next section we give details on the ILS implementations we used in our study. Then, Section 4 gives the results of the analysis of the RTDs. Based on this analysis we present computational results for improved ILS variants in Section 5 and end with some concluding remarks in Section 6.

2 The Travelling Salesman Problem

The TSP is an $\mathcal{NP}$-hard [5] optimization problem which is extensively studied in the literature [11, 13, 22]. It has become a standard test-bed for new algorithmic ideas and a good performance on the TSP is often taken as a proof for the usefulness of an algorithmic approach. Intuitively, the TSP is the problem of a salesman who wants to find, starting from his home town, a shortest possible trip through a given set of customer cities and to return to its home town. More formally, it can be represented by a complete, weighted graph $G = (N, A, d)$ with N being the set of nodes, also called cities, A being the set of edges fully connecting the nodes, and d is a weight function which assigns to each arc $(i, j) \in A$ a value d_{ij} which represents the distance between cities i and j. The TSP is the problem of finding a minimal length Hamiltonian circuit of the graph, where a Hamiltonian circuit is a closed tour visiting each of the $n = |N|$ nodes of G exactly once. For symmetric TSPs, the distances between the cities are independent of the direction of traversing the arcs, that is, $d_{ij} = d_{ji}$ for every pair of nodes. In the asymmetric TSP (ATSP) at least for one pair of nodes i, j we have $d_{ij} \neq d_{ji}$. All the TSP instances used in the empirical studies presented in this article are taken from the TSPLIB Benchmark library accessible at `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html`. These instances are actually Euclidean instances, that is, the cities are given points in the Euclidean space and the distance is the Euclidean distance between the points. These instances, in fact, have been used in many other algorithmic studies and partly stem from practical applications to the TSP. In Figure 2.1 we show two example TSPLIB instances.



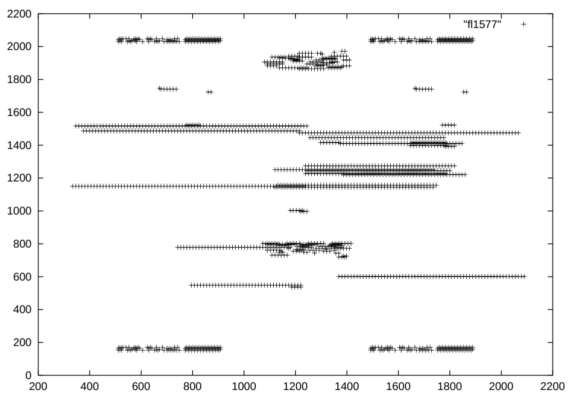


Figure 2.1: TSP instances `att532` (left side) and `fl1577` (right side) from TSPLIB with 532 and 1577 cities, resepctively. The first instance is based in 532 cities in the USA while the second instance, which stems from a drilling problem, shows a pathological clustering of the cities.

3 Iterated Local Search for the TSP

3.1 Algorithmic Outline

Iterated Local Search is a very simple and powerful metaheuristic which has provided the basis for some of the best performing approximation algorithms for the TSP [16, 11]. ILS is based on the observation that local search algorithms are easily trapped in local minima. Instead of restarting the local search from a new, randomly generated solution, it is usually a better idea to modify the current solution s, such that it is moved to a point s' beyond the neighborhood searched by the local search algorithm and to restart the local search from s' to reach a new local minimum s''. An acceptance criterion determines whether the search is continued from s' or s'' (or possibly from some other solution). This is captured in the algorithmic outline given in the following algorithmic outline.

```
procedure Iterated Local Search
    $s_0$ = GenerateInitialSolution
    $s$ = LocalSearch($s_0$)
    repeat
        $s'$ = Modify($s$)
        $s''$ = LocalSearch($s'$)
        $s$ = AcceptanceCriterion($s, s''$)
    until termination condition met
end
```

The instantiation of the functions LocalSearch, Modify, and AcceptanceCriterion is an important issue in the design of ILS algorithms. In the following we discuss the choices for these functions for the ILS algorithms we applied to the TSP.

3.2 Generic Algorithm Choices for the Symmetric TSP

3.2.a Choice of LocalSearch

In principle, any local search algorithm can be used, but the choice of the particular local search algorithm may have a crucial influence on the final performance of the ILS algorithm. In this paper we consider three possible local search algorithms for the TSP. The most simple ones to implement are `2-opt` and `3-opt`, which try to find an improved solution by exchanging two or at most three edges, respectively. However, typically the best performance with respect to solution quality is obtained with the considerably more complex Lin-Kernighan heuristic (`LK`) [14]. For each move, it considers a variable number of edges to be exchanged. Yet, a disadvantage of the `LK` heuristic is that considerable implementation effort is required to achieve short run-times and best performance with respect to solution quality. `2-opt` and `3-opt`, on the other hand, are much simpler to implement and may therefore be the preferred choice for practitioners. Additionally, `2-opt` and `3-opt`'s run-time is much more robust with respect to the particular instance type than `LK`'s run-time. In particular, `LK`'s run-time grows strongly when applied to highly clustered TSP instances like instance `fl1577` plotted in Figure 2.1 [11]. Therefore, we did not apply the `LK` heuristic to such clustered instances. In the following, we will refer to the ILS algorithms using `2-opt` and `3-opt` and `LK` local search as Iterated `2-opt` (`I2opt`), Iterated `3-opt` (`I3opt`), and Iterated `LK` (`ILK`), respectively.

In the experiments with ILS we used our own `2-opt` and `3-opt` implementation, while the `LK` local search algorithm was kindly provided by Olivier Martin (details on that implementation are given in [16]). Our `2-opt` and `3-opt` implementations use standard speed-up techniques which are described in [3, 11, 17]. In particular we perform a fixed radius nearest neighbor search within candidate lists of the 40 nearest neighbors for each city and use don't look bits. Initially, all don't look bits are turned off (set to 0). If for a node no improving move can be found, its don't look bit is turned on (set to 1) and the node is not considered as a starting node for finding an improving move in the next iteration. When an arc incident to a node is changed by a move, the node's don't look bit is turned off again. These speed-up techniques achieve that both algorithm's run-time grows subquadratically with instance size.

3.2.b Choice of Modify

For Modify we use the so called *double-bridge* move which is the standard choice for ILS applications to the TSP [17, 16, 11]. It cuts the current tour at four appropriately chosen edges into four subtours $s_1 - s_2 - s_3 - s_4$ which are contained in the solution in the given order. These subtours are then reconnected in the order $s_4 - s_3 - s_2 - s_1$ to yield a new starting solution for the local search. The double-bridge move is a specific 4-opt move done in such a way that it cannot be reversed directly by either of the three local search algorithms applied here. In preliminary experiments we found that best performance is obtained if the edges cut by the double-bridge move are not chosen completely at random. Instead, we proceed as follows: in `I2opt` and `I3opt` we first randomly choose an edge (i, j) at random. The other three cut-points are then chosen such that they all lie within a candidate set of the $\max\{n/2, 500\}$ nearest neighbors of city i. Such an approach has also been shown to be effective on large TSP instances [23]. Alternatively, length restrictions on the newly introduced edges are imposed in the original `ILK` code of [17]. In particular, in the experiments we report here a newly introduced edge has to be shorter than 25 times the average edge length of the current solution.

We further found that the interaction between the double-bridge move and the setting of the don't look bits after Modify is important to achieve a good tradeoff between solution quality and run-time. For `ILK` it was found experimentally was found that a very good tradeoff is achieved by only resetting only the don't look bits of the cities directly affected by the double-bridge move to zero [11, 18]. Yet, for `I2opt` and `I3opt` we found that the performance with such a resetting strategy was relatively poor with respect to solution quality. Here, a good compromise was found by resetting the don't look bits of a number of the cities, in the experiments chosen as 20, preceding and following a cut-point.

3.2.c Choice of AcceptanceCriterion

Common knowledge in ILS applications to the TSP and also to other combinatorial optimization problems appears to be that accepting better quality solutions only gives the best performance [15, 16, 10, 11, 7, 6]. Consequently, we will use this acceptance criterion as the basis of our analysis; we will denote it as $Better(s, s'')$ and it returns s'' if tour s'' is shorter than s, otherwise it returns s. If side-moves are allowed, that is, a new solution is also accepted if it has the same cost as the current one, we will call the acceptance criterion $BetterEqual(s, s'')$. Yet, it should already be noted here that only occasionally for specific instances an improved performance has been reported with acceptance criteria which may accept worse solutions with a small probability [17, 23].

3.2.d Initial Solution

The initial solution for all the ILS algorithms applied are generated by the nearest neighbor heuristic. Compared to random initial tours, this choice reduces the run-time for the first local search application.

3.3 Generic Algorithm Choices for the Asymmetric TSP

When applying ILS to the ATSP, the generic algorithm choices are the same as for the symmetric TSP except that we apply a particular `3-opt` algorithm which we called `reduced 3-opt`. Note that in `2-opt` and some `3-opt` moves subtours have to be traversed in the opposite direction and in this case for the ATSP the length of that subtour would have to be re-computed from scratch. To avoid this, `reduced 3-opt` only applies one specific `3-opt` move which does not lead to a reversal of any subtour. For modify, again the double-bridge move can be applied since it does not reverse any subtour. The acceptance criterion and the choice of the initial solution are the same as for the symmetric TSP.

Surprisingly, in the literature no results for the application of ILS to the ATSP are reported. The only ILS application to ATSPs is described in [26] in the context of a specific code optimization problem which is formulated as an ATSP. But there, the ATSP instances are solved via a transformation to the symmetric TSP and by then applying an iterated `3-opt` algorithm for symmetric TSPs. Consequently, the results presented here are the first evidence that ILS can be applied directly to the ATSP with considerable success.

4 Run-Time Distributions for ILS Applied to the TSP

In this section we analyze the empirical run-time behavior of iterated local search algorithms for the TSP. In particular, we study RTDs for `I2opt`, `I3opt`, and `ILK` for the symmetric TSP and `Ired3-opt` for the ATSP. Since the run times needed to observe limiting behavior of the algorithms become rather high for large TSP instances, we limited most part of the study to instances with less than 1000 cities. For each instance, RTDs have been measured using 100 independent runs of each algorithm and in the plots always on the x-axis the run-time is indicated and on the y-axis the cumulative empirical RTD is given.

The run-time distributions are given with respect to various bounds on the solution quality. In addition to the empirical run-time distributions we plot the cumulative distribution function of an exponential distribution (indicated by $f(x)$ in the plots) adjusted towards a run-time distribution corresponding to high quality solutions. These particular exponential functions (the distribution function is given by $f(x) = 1 - e^{-\lambda x}$, $x \geq 0$) give valuable hints on possible improvements of an algorithm's performance. Recall that, based on a well-known theorem from probability theory, if a given algorithm has an exponential RTD, then the probability of finding a solution by running the algorithm k times for time t is the same as when running the algorithm once for time $k \cdot t$. Hence, if the RTD is in fact an exponential, in the long run, such restarts will not affect the solution probability. The plotted exponential distributions give an indication whether the algorithm can be improved by restarting it from a new initial solution. This is the case if the empirical run-time distribution falls below the plotted exponential. As we will see, for the ILS algorithms considered here, this occurs for many of the instances.

To obtain these run-time distributions we limited the maximally allowed computation time on the single instances to some upper CPU-time limit which was increased with increasing instance size. In each case the computation times are chosen large enough to ensure that the algorithm shows asymptotic performance and that the further increase of solution quality by allowing still longer computation times should be negligibly small.

4.1 Operation counts

Actual run-time distributions may be obtained by directly measuring CPU-time or by using representative operation counts as a more machine independent measure of an algorithm's performance

[1]. A natural choice for a high-level operation count for the ILS algorithms applied here is given by the number of times a local search is applied.[1] To establish the relation between CPU-time and local search iterations, in Table 4.1 we give the average CPU-time measured on a single 266MHz Pentium II CPU and 320MB RAM under Redhat Linux 5.2 when running `I2opt`, `I3opt`, and `Ired3-opt` for 25,000 iterations and `ILK` for 1,000 iterations on the instances used in this study. The actual times are averaged over 5 independent ILS runs on each instance. When measuring the run-time based on operation counts, we will call the resulting distributions also run-length distributions (RLDs) in the following.

For most of our experiments, we report actual CPU-times measured either on a single 167MHz UltraSparc I processor and 192MB RAM or the 266MHz Pentium II processor on which the timings in Table 4.1 are based. Comparing the computation times on the two machines we found that the Pentium II was roughly 3.2 times faster than the UltraSparc I processor.

Table 4.1: CPU-time taken for running `I2opt`, `I3opt`, and `Ired3-opt` on a 266MHz Pentium II CPU for 25,000 iterations. Run-times are averaged over 5 independent runs of the ILS algorithms.

					`I2opt`						
`eil51`	3.07	`kroA100`	5.39	`lin105`	5.43	`u159`	7.07	`d198`	9.10	`lin318`	13.25
`pcb442`	16.59	`rat783`	27.94	`pr1002`	34.66	`pcb1173`	39.42	`d1291`	41.17	`fl1577`	46.96
					`I3opt`						
`d198`	68.0	`lin318`	97.2	`pcb442`	64.4	`rat783`	118.5	`pr1002`	145.2	`pcb1173`	140.0
`d1291`	113.9	`fl1577`	124.8	`pr2392`	180.1	`pcb3038`	209.7	`fl3795`	208.3		
					`ILK`						
`lin318`	714.7	`pcb442`	577.9	`att532`	433.42	`rat783`	163.5	`pcb1173`	337.8	`pr2392`	482.44
					`reduced 3-opt`						
`ry48p`	27.8	`ft70`	53.8	`kro124p`	50.4	`ftv170`	58.1	`rbg443`	285.6		

4.2 Run-time Distributions for Symmetric TSPs with Iterated 2-opt

Figure 4.1 shows the empirically observed RTDs for the `I2opt` algorithm on the instances four TSPLIB instances `d198`, `lin318`, `pcb442`, and `rat783`. Except for instance `d198` `I2opt` could only very rarely find the optimal solutions for these instances within reasonable run-time. Thus, we report mainly RTDs which are obtained by imposing some weaker bounds on the solutions quality. In the plots, the exponential distribution is fitted to the lower part of run-time distributions corresponding to high quality solutions (where high quality is here meant as high relative to `I2opt`). As can be observed in the plots, the exponential matches the lower part of the RTDs well, yet, for larger run-times the RTDs fall strongly below the indicated exponential. Hence, the `I2opt` algorithm is severely affected by stagnation behavior and by using restarts, the solution probability can be significantly improved. We will give a more detailed example of this stagnation behavior in the next section when examining RTDs for `I3opt`.

4.3 Run-time Distributions for Symmetric TSPs with Iterated 3-opt

Figure 4.2 shows the empirical run-time distributions for the four TSPLIB instances `d198`, `lin318`, `pcb442`, and `rat783` are given in Figure 4.2. Additionally, exponential distributions which may indicate stagnation behavior are approximated to RTDs for high-quality solutions.

[1]Note that our implementation choices for `2-opt` and `3-opt` were made in such a way that a very large number of local search runs can be done in a given amount of CPU-time. Other implementation choices may lead to a somewhat more effective local search in terms of solution quality but may compromise the speed of the local search. One example of such a choice is the particular resetting strategy of the don't look bits in `I2opt` and `I3opt`.

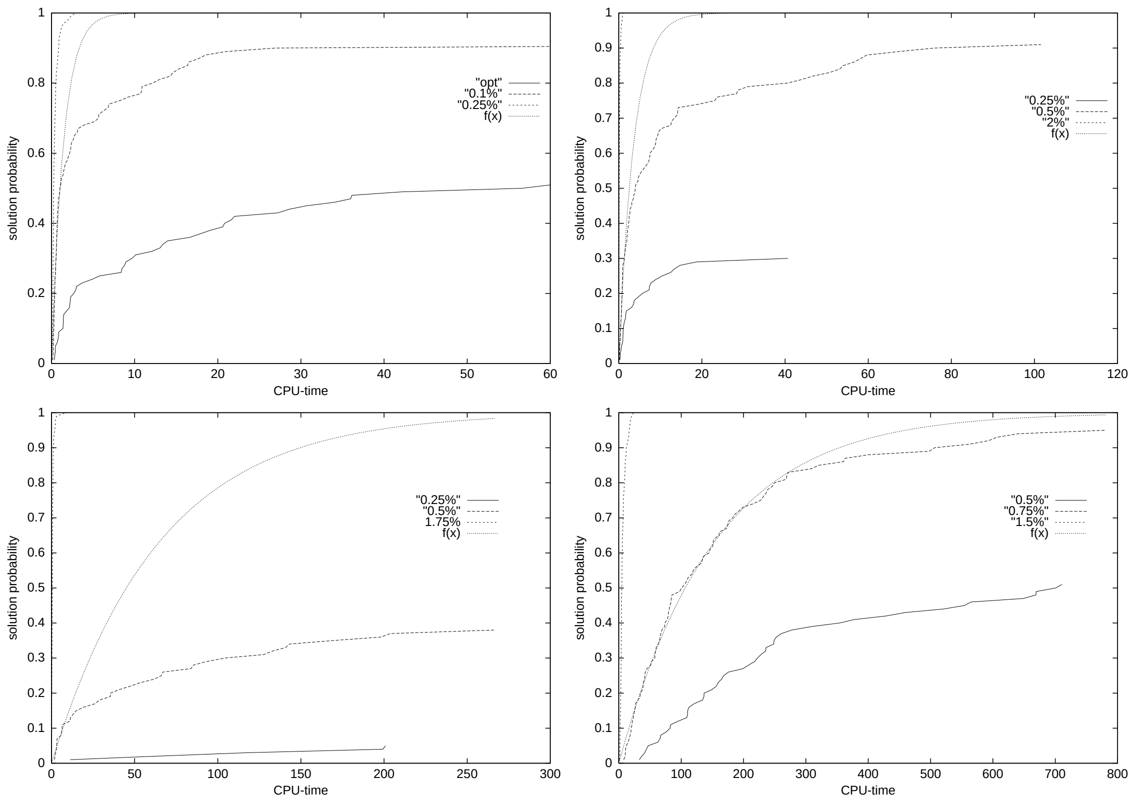


Figure 4.1: Run-time distributions of iterated `2-opt` on four symmetric TSP instances. RTDs are identified by the required solution quality (e.g., opt means optimal solution is required). Given are the distributions for `d198` (upper left side), `lin318` (upper right side), `pcb442` (lower left side), and `rat783` (lower right side). $f(x)$ indicates an exponential distribution. See the text for more details.

From the empirical run-time distributions several conclusions can be drawn. One is that on all instances it is rather easy to get within, ca. 0.5% or 1% (in the case of `pcb442`), of the optimum; the empirical run-time distribution corresponding to one of these bounds are always the leftmost ones in the plots. Yet, if higher accuracy is required, the algorithms suffer from stagnation behavior. Consider, for example, instance `pcb442`. If we require a solution quality bound of 0.5% within the global optimum, after only 15 seconds an empirical solution probability of 0.89 is obtained (runs were performed on the UltraSparc I processor). Yet, in 4 of the 100 trials the algorithm fails to reach the required solution quality bound even after 800 seconds. A similar situation occurs for tighter solution quality bounds, like when searching for optimal solutions. For `pcb442` in some runs optimal solutions can be found very fast, reaching a solution probability of 0.39 after 60 seconds, but after 940 additional seconds the solution probability only reaches 0.62.[2] Similar observations also apply to the other instances. This stagnation behavior is especially striking on the instances `lin318`, `pcb442`, and `rat783`. On instance `d198` the run-time distribution for finding the optimal solution is actually close to an exponential distribution, although for larger run-times it is slightly less steep. Since `I2opt` has shown a strong stagnation behavior on this latter instance, we may conclude that the more powerful `3-opt` local search is responsible for the better behavior on that instance.

[2]Note that, when restarting `I3opt` on this instance after 60 seconds, with 10 restarts (that is, 600 seconds) one would estimate to obtain a solution probability of roughly 0.993 which is much larger than the empirically observed solution probability of 0.62 without using restart.

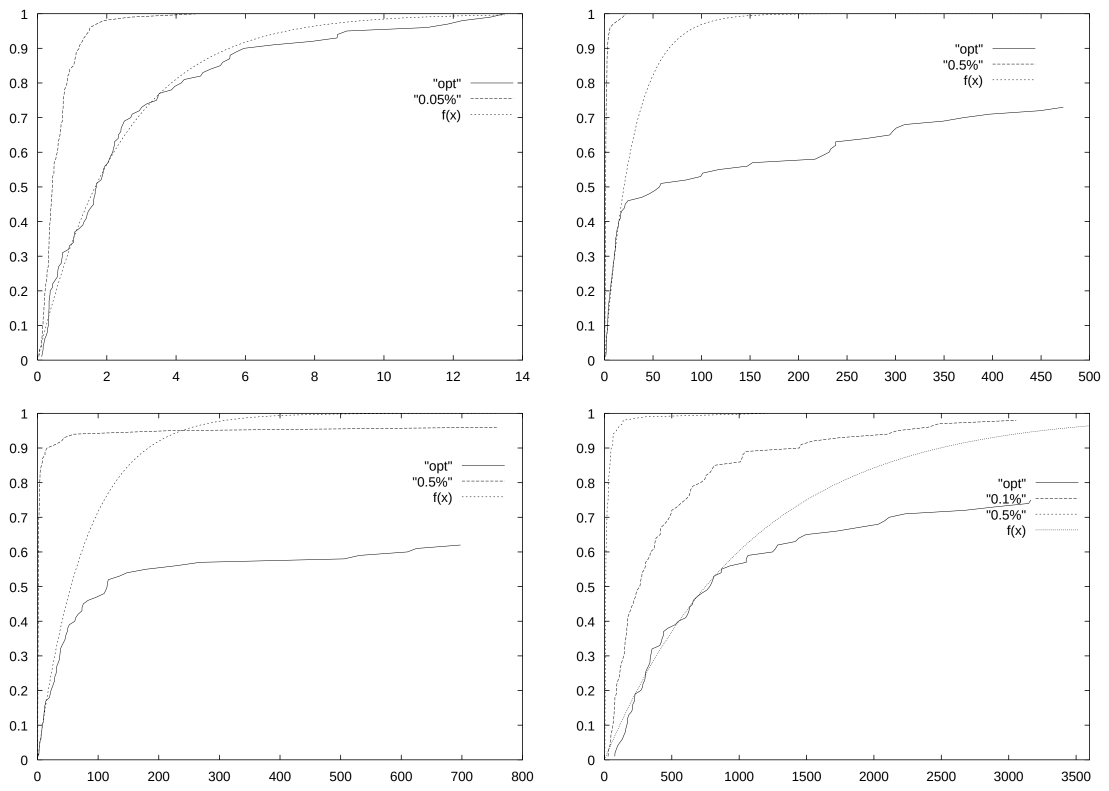


Figure 4.2: Run-time distributions of iterated `3-opt` on four symmetric TSP instances. RTDs are identified by the required solution quality (e.g., opt means optimal solution is required). Given are the distributions for `d198` (upper left side), `lin318` (upper right side), `pcb442` (lower left side), and `rat783` (lower right side).

4.4 Run-time Distributions for Symmetric TSPs with `ILK`

One could object that the observation of a stagnation behavior in general may be due to the choice of the `2-opt` or `3-opt` local search algorithms which shows worse performance on the TSP than the more sophisticated `LK` heuristic. To test this hypothesis we have ran the `ILK` heuristic on the same instances used in the previous section, except the smallest one which was easily solved. The analysis presented in this section is based on measuring run-*length* distributions (RLDs), where we use the number of `LK` applications as operation counts.

Since `ILK` is known to be one of the best performing approximate algorithms for symmetric TSP instances, we expect higher quality solutions to be obtained more frequently than with either `2-opt` or `3-opt`. This expectation is confirmed by the empirical run-length distributions given in Figure 4.3.[3] For the instances `pcb442` and `rat783` the empirical probability of finding an optimal solution is significantly higher than for `I3opt`. Instances `pcb442` and `rat783` are solved in every trial to optimality by the `ILK` algorithm. In particular, the run-length distribution of `pcb442` can be closely fitted by a modified exponential distribution with distribution function $G(x) = \max\{0, 1 - e^{-\lambda(x-\Delta x)}\}$, where Δx adjusts for the fact that a certain minimal number of iterations have to be to performed to obtain a

[3]Instance `lin318` is not shown here since it was solved to optimality in all runs. On instance `att532` `I3opt` showed a similar strong stagnation behavior as `ILK`, yet the final solution probabilities for reaching the optimal solution were significantly lower.

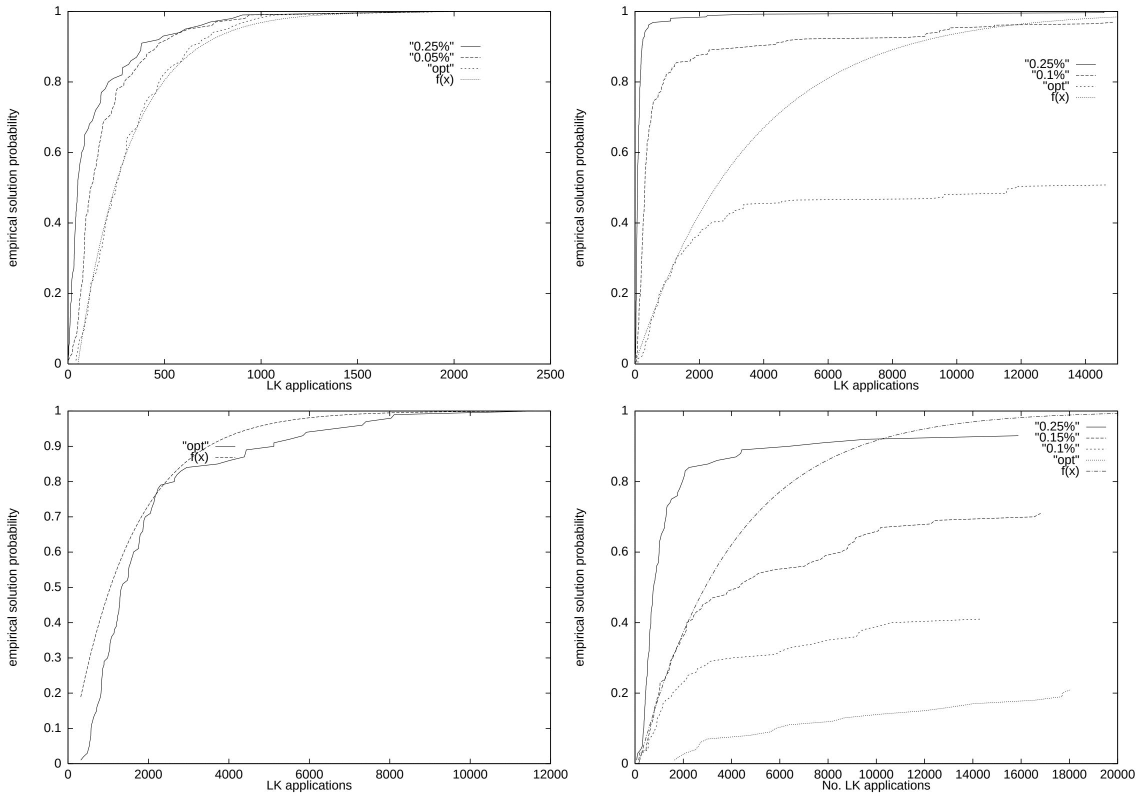


Figure 4.3: Run-length distributions of `ILK` on four symmetric TSP instances. RTDs are identified by the required solution quality (e.g., opt means optimal solution is required). Given are the distributions for `pcb442` (upper left side), `att532` (upper right side), `rat783` (lower left side), and `pr2392` (lower right side).

reasonable chance of finding the optimal solution.[4] In general, it appears that this particular instance is easily solved using the `LK` heuristic, but it is "relatively hard" to solve when using `2-opt` or `3-opt` (compare the run-time distribution of `ILK` to that of iterated `3-opt` in Figure 4.2). On instance `rat783`, `ILK` again falls slightly below the exponential distribution, which indicates stagnation behavior and suggests that the algorithm's performance could possibly be further improved. On the other two instances `att532` and `pr2392` again the stagnation behavior is clearly visible.

4.5 Run-time Distributions for ATSPs with Iterated `reduced 3-opt`

The run-time distributions for the ATSP instances shown in Figure 4.4 confirm the findings on the symmetric instances. On the ATSP instances the stagnation behavior of iterated `reduced 3-opt` appears to be even more severe. Often, the optimal solution is obtained very early in a run or, with a few exceptions, is not found at all within the given computation time. Surprisingly, also on the smallest instance `ry48p` with only 48 nodes, the algorithm shows stagnation behavior; however, with larger run-time it appears that all runs could be terminated successfully.

Interestingly, despite its size, instance `rbg443`, with 443 cities the largest ATSP instance of

[4]The curve in that plot has been fitted with the C. Grammes implementation of the Marquart-Levenberg algorithm available in gnuplot. The fitted parameters are $\Delta x = 51.26$ and $\lambda = 0.00524$.

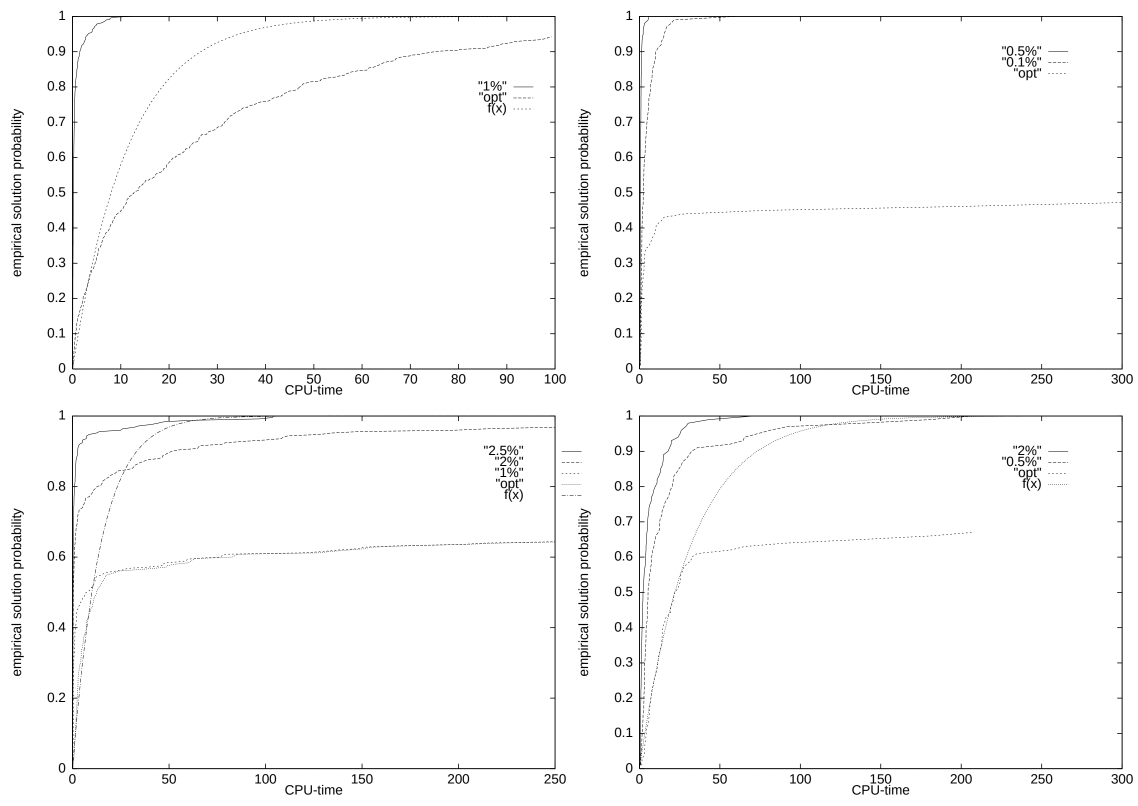


Figure 4.4 : Run-time distributions of iterated `reduced 3-opt` on four ATSP instances. RTDs are identified by the required solution quality (e.g., opt means optimal solution is required). Given are the run-time distributions for `ry48p` (upper left side), `ft70` (upper right side), `kro124p` (lower left side) and `ftv170` (lower right side).

TSPLIB, is easily solved if the acceptance criterion $BetterEqual(s, s'')$ is used. In this case an optimal solution is found, on average, in 35 seconds or 5200 local search applications, with the maximally required run-time to solve this instance being roughly 150 seconds. The same instance could only be solved in 90% of the runs with maximally 1000 seconds if the acceptance criterion $Better(s, s'')$ is used. In Figure 4.5 we have plotted the run-time distributions for these two acceptance criteria. Such significant differences between these two acceptance criteria could only be observed on the four ATSP instances `rbg323`, `rbg358`, `rbg403`, and `rbg443`. Considering this fact, in the later experimental investigation we used the acceptance criterion $BetterEqual(s, s'')$ for all ATSP instances.

4.6 Observations from the Run-Time Distributions

A general conclusion from the RTD-based analysis of ILS algorithms is that these algorithms tend to quickly find good solutions to symmetric as well as asymmetric TSPs. But if very high solution quality is required, one could observe using the RTD-plots that the ILS algorithms suffer from search stagnation which severly compromises ILS performance. The empirical RTDs are, in fact, in nearly all cases less steep than an exponential distribution which well approximates the run-time distribution in the lower part — and therefore the algorithm can be improved (in the simplest case) by introducing occasional restarts after a fixed number of iterations (cutoff time). By restarting the algorithm after an appropriately determined cutoff time, one actually forces the ILS algorithm's solution probabilty

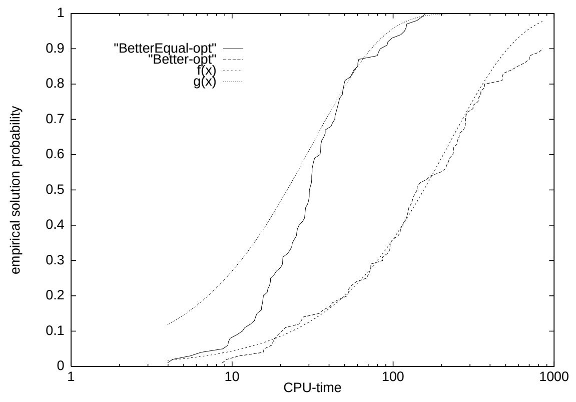


Figure 4.5 : Run-time distributions for instance `rbg443` using iterated `reduced 3-opt`. Given are two run-time distribution, one using acceptance criterion $BetterEqual(s, s'')$, the other using $Better(s, s'')$. Additionally two exponential distributions approximate each of the two empirical RTDs (indicated by $f(x)$ and $g(x)$).

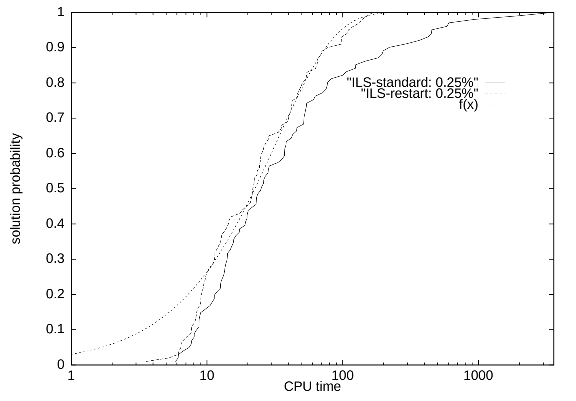


Figure 4.6 : Example of improvements by using restart. Given is the run-time distribution for reaching a solution within 0.25% of the optimum on instance `att532` for `I3opt` (indicated by ILS-standard) and an ILS algorithm using restart after an appropriately chosen cutoff time (indicated by ILS-restart). The improved ILS algorithm is able to follow the exponential distribution (indicated by f(x)) which approximates ILS-standard's RTD in the left part. See text for more details.

distribution to follow the exponential distributions. This fact is exemplified in Figure 4.6 for an `I3opt` algorithm applied to instance `att532`.

Our investigation of the run-time distribution also shows that the more effective is the local search algorithm, the less is affected the ILS algorithm from the stagnation behavior. For example, the `I2opt` algorithm showed a very strong stagnation behavior on instance `d198` when trying to find the optimal solution, while `I3opt`'s RTD followed reasonably well an exponential distribution. On the larger `pcb442` instance both, `I2opt` and `I3opt`, were severely affected by the stagnation behavior, while `ILK` could solve this instance rather easily. Yet, when applied to larger instances, `ILK` again shows stagnation behavior and could be considerably improved. Hence, one possible conclusion from these observations is that the ILS algorithms will — when applied to non-trivial instances — show stagnation behavior from a specific instance size. This instance size depends on the effectiveness of the local search algorithm.

In summary, the run-time distributions arising from the application of ILS algorithms to the TSP suggest that these algorithms can be further improved. In the following we will present several possibilities how such improvements can be achieved, followed by an empirical investigation of these new ILS variants.

5 Improvements of ILS Algorithms for the TSP

In this section we will present two improved ILS algorithms for the TSP and then give results from an extensive experimental comparison of these algorithms.

5.1 Improved ILS algorithms

One conclusion from the observed stagnation behavior of ILS is that additional diversification features are needed to increase ILS's performance. There are two particular ways to increase the diversification in ILS algorithms. A first possibility is to use acceptance criteria which allow moves to worse solutions or, as proposed before, one may restart the algorithm. A second possibility is to increase the strength of the solution modifications implemented by Modify; for the TSP this can be done by cutting the current tour at more than four points, as it is done in the doublebridge move, or vary the strenght of the solution modifications as it is also proposed in basic Variable Neighborhood Search (basic VNS) [7].

Here, we propose and experimentally investigate two improved ILS variants based on the first possibility which is also directly motivated by our RTD-based analysis of ILS behavior.

5.1.a Soft Restarts

As argued before, the simplest strategy to avoid stagnation behavior is to restart the algorithm from new initial solutions after some predefined cutoff value. Yet, optimal cutoff values may depend strongly on the particular TSP instance. A better idea appears to be best to relate the restart to the search progress and to apply *soft* cutoff criteria. In particular, we restart the ILS algorithm if no improved solution could be found for i_r iterations, where i_r is a parameter. The soft restart criterion is based on the assumption that after i_r iterations without improvement the algorithm is stuck and that additional diversification (implemented by restarting the algorithm from a new initial solution) is needed to escape from there. The restart of the algorithm can be easily modeled by the acceptance criterion called $\mathit{Restart}(s, s'', \mathit{history})$ (the *history* component indicates that the use of soft restarts can be interpreted as a very simple use of the search history). Let i_{last} be the last iteration in which a better solution has been found and i be the iteration counter. Then $\mathit{Restart}(s, s'', \mathit{history})$ is defined as

$$\mathit{Restart}(s, s'', \mathit{history}) = \begin{cases} s'' & \text{if } f(s'') < f(s) \\ s''' & \text{if } f(s'') \geq f(s) \text{ and } i - i_{last} > i_r \\ s & \text{otherwise,} \end{cases} \qquad (.1)$$

where s''' can be a randomly generated new initial solution.

5.1.b Fitness-Distance Diversification-based ILS

A disadvantage of restarting ILS from new initial solutions is that previously obtained high quality solutions are lost. Additionally, ILS algorithms need some initialization time t_{init}, which will increase with increasing instance size, for finding very high quality solutions. Hence, with each restart a CPU-time t_{init} is wasted. To avoid these disadvantages, a better choice may be to use a more directed diversification when the ILS algorithm is supposed to be stuck. Consequently, we propose a new method to escape from the current search space region. Our goal will be to *find a good quality solution beyond a certain minimal distance from the current search point without using restart*. In fact there are two objectives to be taken into account to find such a solution. The first is to generate a distant solution, the second is that this solution should be of reasonable quality.

We implemented this idea as follows. Let s_c be the solution from which we want to escape and the distance between solutions is measured as the number of different edges. Then the following steps are repeated until we have found a solution beyond a minimal distance $d_{\min}$ from s_c.

(1) Generate an initial population of p copies of s_c.

(2) To each solution, first apply Modify followed by an application of LocalSearch.

(3) Choose the best q of the p solutions, $1 \leq q \leq p$, as candidate solutions.

(4) Let s be the candidate solution with maximal distance to s_c

(5) If the distance between s and s_c is smaller than $d_{\min}$, then repeat at (2); otherwise return the candidate solution at maximal distance.

The single steps are motivated by the following considerations. The second objective (find a good quality solution) is attained by keeping only the q best of the p candidate solutions. The first objective (find a solution beyond a minimal distance to s_c) is achieved by choosing a candidate solution with maximal distance from s_c. The steps are then iterated until a solution s^* is found for which the distance requirement $d_{\min}$ is satisfied. In fact, we stop this process if we have not found a solution s^* beyond the required minimal distance after a maximal number of iterations (here 30) through these steps.

5.2 Experimental Analysis

This section presents the results of an experimental comparison of three ILS algorithms, the standard ILS algorithm used for the run-time analysis, the ILS algorithm using soft restarts (referred to as ILS-restart), and the ILS algorithm with fitness-distance based diversification (referred to as ILS-FDD). In ILS-restart and ILS-FDD a restart (diversification) is initiated if for i_r iterations no improved solution is found, where $i_r = 3 \cdot n$ for `I2opt`, $i_r = n$ for `I3opt`, and finally $i_r = 1/3n$ for `ILK`; n is the number of cities of a TSP instance. For ILS-FDD we use $p = 20$, $q = 15$ (parameters were chosen in an ad hoc manner without fine-tuning). Since an appropriate value for $d_{\min}$ may be instance dependent, we estimate $d_{\min}$ by the average distance d_{avg} between 100 (50 when applying `LK`) locally optimal solutions and then setting alternatingly $d_{\min} = 1/4 \cdot d_{avg}$ and $d_{\min} = 1/2 \cdot d_{avg}$. In ILS-FDD diversification is always applied to the best solution found since the start of the algorithm (only when using `LK` this is done every second diversification). This choice is motivated by results on the TSP search space analysis where it was shown that the better a solution, the closer it is on average to a global optimum [4].

The computational results are given in Table 5.2 to Table 5.4 for the ILS algorithms based `2-opt`, `3-opt`, and `LK`, respectively and in Table 5.5 for the ILS application to the ATSP. In each table are

Table 5.2: Comparison of `I2opt`, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance (the number in the instance identifier is the problem size), we report how often the known optimal solution is found in a given number of runs (n_{opt}/no. trials), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max}. The algorithms were run on a 266MHz Pentium II CPU with 320 MB RAM running Redhat Linux.

Instance	opt	ILS `2-opt`			ILS-Restart			ILS-FDD			t_{max}
		n_{opt}	avg	t_{avg}	n_{opt}	avg	t_{avg}	n_{opt}	avg	t_{avg}	
`eil51`	426	20/100	0.19	< 0.1	100/100	0.0	0.4	100/100	0.0	0.2	20
`kroA100`	21282	100/100	0.0	< 0.1	100/100	0.0	< 0.1	100/100	0.0	0.2	60
`lin105`	14379	100/100	0.0	< 0.1	100/100	0.0	< 0.1	100/100	0.0	0.2	60
`u159`	42080	100/100	0.0	7.3	100/100	0.0	0.4	100/100	0.0	0.3	60
`d198`	15780	59/100	0.032	18.5	100/100	0.0	18.8	100/100	0.0	9.4	120
`lin318`	42029	3/100	0.30	15.9	21/100	0.11	51.8	94/100	0.008	37.8	120
`pcb442`	50778	1/100	0.62	118.6	0/100	0.31	131.2	5/100	0.15	185.2	300
`rat783`	8806	0/100	0.49	536.2	0/100	0.60	415.0	0/100	0.29	613.9	900
`pr1002`	259045	0/25	0.55	922.7	0/25	0.81	750.2	0/25	0.25	974.6	1200
`pcb1173`	56892	0/25	1.10	787.3	0/25	0.90	693.5	0/25	0.54	918.3	1200
`d1291`	50801	0/25	0.68	523.4	0/25	0.27	712.4	0/25	0.12	789.8	1200
`fl577`	22249	0/25	0.82	996.6	0/25	0.11	1262.0	0/25	0.05	1463.1	2400

given the number of optimal solutions found by an algorithm in a given number of independent trials, the average percentage deviation from the optimum and the average run-time to find the best solution in each trial. Additionally, we indicate the maximal run-time t_{max} allowed for an algorithm. t_{max} was chosen in roughly such a way that larger instances are also given increasing run-time.

In general, the experimental results show that the performance of all ILS algorithms can significantly be improved with respect to solution quality. With the two ILS extensions the frequency of finding optimal solutions is strongly increased and the average solution quality improves considerably. The only exception are the results obtained with ILS-restart using `2-opt` on the instances `rat783` and `pr1002`. Since at least instance `rat783` has shown apparent stagnation behavior in Figure 4.1 when using `I2opt`, most probably the parameter i_r was chosen too low. Yet, notice that also for these two instances ILS-FDD significantly improves over standard `I2opt`. Similarly, when applying `I3opt`, for `rat783` the frequency of finding the optimum is lower with ILS-restart than with `I3opt`. Regarding the ATSPs, all instances showing stagnation behavior are well solved already by using ILS-restart (see Table 5.5) and therefore we did not apply ILS-FDD on these instances since apparently they do not provide a real challenge.

The relative performance of the three ILS algorithms depends strongly on the instance size. In particular, for small problems ILS-restart and ILS-FDD perform very similarly to standard ILS if the latter does not show any or only very weak stagnation behavior (notice that `I2opt` shows very strong stagnation behavior on instance `eil51` which is the smallest instance we tested). Yet, with increasing instance size the performance improvement obtained with ILS-restart and ILS-FDD becomes very obvious.

Similarly, the performance advantage of ILS-FDD over ILS-restart becomes more marked for larger instances. ILS-FDD shows on all instances with more than 400 cities either better average performance and a higher frequency of finding the optimal solution or, if both algorithms find the optimal solution in all runs, a much lower average run-time. It should also be noted that the performance of ILS-FDD using `3-opt` local search is particularly good on instances which are known to be hard for other algorithms. This is the case for `fl1577` and `fl3795` which are highly clustered (as shown in Figure 2.1 on the right side). On such instances also ILS-restart performs surprisingly good. This is probably due to the fact that these instances contain deep local minima from which the ILS algorithm has strong problems to escape but which are overcome by restarting the algorithm from a new initial

Table 5.3: Comparison of `I3opt`, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance (the number in the instance identifier is the problem size), we report how often the known optimal solution is found in a given number of runs (n_{opt}/no. trials), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max}. The algorithms were run on a 266MHz Pentium II CPU with 320 MB RAM running Redhat Linux.

Instance	opt	ILS `3-opt`			ILS-Restart			ILS-FDD			t_{max}
		n_{opt}	average	t_{avg}	n_{opt}	average	t_{avg}	n_{opt}	average	t_{avg}	
`d198`	15780	100/100	0.0	1.1	100/100	0.0	0.8	100/100	0.0	1.5	120
`lin318`	42029	65/100	0.10	13.9	100/100	0.0	7.1	100/100	0.0	13.7	120
`pcb442`	50778	56/100	0.12	34.9	100/100	0.0	46.5	100/100	0.0	30.8	300
`att532`	27686	22/100	0.055	91.6	74/100	0.0096	214.6	96/100	0.002	202.3	600
`rat783`	8806	71/100	0.029	238.8	51/100	0.018	384.9	100/100	0.0	159.5	900
`pr1002`	259045	14/25	0.11	389.7	25/25	0.0	578.2	25/25	0.0	207.2	1200
`pcb1173`	56892	0/25	0.26	461.4	0/25	0.040	680.2	14/25	0.011	652.9	1200
`d1291`	50801	2/25	0.29	191.2	17/25	0.012	410.8	25/25	0.0	245.4	1200
`fl577`	22249	3/25	0.52	494.6	23/25	0.00008	477.6	25/25	0.0	294.1	2400
`pr2392`	378032	0/10	0.23	1538.5	0/10	0.22	2008.5	3/10	0.027	2909.1	3600
`pcb3038`	137694	0/10	0.22	3687.6	0/10	0.20	4824.3	0/10	0.099	5535.9	7200
`fl3795`	28772	2/10	0.36	3601.9	2/10	0.0035	3080.9	9/10	0.0003	3506.7	7200

Table 5.4: Comparison of `ILK`, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance, we report how often the known optimal solution is found in a given number of trials (n_{opt}/no. trials), the average percentage deviation from the optimum, the average number of `LK` applications i_{avg} to find the best solution in a run, and the maximally allowed number of `LK` applications.

Instance	opt	`ILK`			ILS-Restart			ILS-FDD			t_{max}
		n_{opt}	average	i_{avg}	n_{opt}	average	i_{avg}	n_{opt}	average	i_{avg}	
`lin318`	42029	25/25	0.0	299.8	25/25	0.0	165.8	100/100	0.0	300.5	5000
`pcb442`	50778	25/25	0.0	325.1	25/25	0.0	289.8	25/25	0.0	297.3	5000
`att532`	27686	17/25	0.029	1834.6	25/25	0.0	3767.5	25/25	0.0	2194.1	10000
`rat783`	8806	24/25	0.01	2087.9	25/25	0.0	3577.2	25/25	0.0	2596.5	10000
`pcb1173`	56892	1/10	0.0065	2071.3	2/10	0.0018	8403.0	8/10	0.0004	5629.9	20000
`pr2392`	378032	1/10	0.11	10924.3	0/10	0.049	9843.8	4/10	0.014	13028.4	25000

Table 5.5: Comparison of `Ired3-opt` and ILS-restart on ATSPs. For each instance, we report how often the known optimal solution is found in a given number of trials (n_{opt}/no. trials), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max}. The algorithms were run on a 167MHz UltraSparc I CPU with 192 MB RAM running Solaris 5.6.

Instance	opt	ILS `3-opt`			ILS-Restart			t_{max}
		n_{opt}	average	t_{avg}	n_{opt}	average	t_{avg}	
`ry48p`	14422	95/100	0.03	24.7	25/25	0.0	9.2	120
`ft70`	38673	14/25	0.056	2.2	25/25	0.0	7.4	300
`kro124p`	36230	17/25	0.056	22.8	25/25	0.0	11.3	300
`ftv170`	2755	23/25	0.10	35.3	25/25	0.0	45.3	300
`rbg443`	2720	25/25	0.0	31.7	25/25	0.0	36.3	900

solution. Also note that on these instances it appears preferable to run `3-opt`, since the run-time of `LK` increases dramatically (see also the more detailed discussion in [11]), while `3-opt` is not affected by this phenomenon.

Recently, several high performing new approaches and improved implementations of known approaches have been presented for the TSP. Among these algorithms we find the genetic local search approach of Merz and Freisleben [18, 19], a new genetic local search approach using a repair-based crossover operator and brood selection by Walters [25], a specialized local search algorithm for the TSP called Iterative Partial Transcription (ITP) by Möbius et.al. [21], and the ILS approach by Katayama and Narisha [12] which uses a newly designed solution modification mechanism inspired by genetic algorithms. The improved ILS algorithms and, in particular, ILS-FDD compare very well to these approaches. When comparing algorithms using only `3-opt` local search, ILS-FDD compares favorably to Walter's GA (that algorithm is run on a 300MHz Pentium II CPU). ILS-FDD achieves, in general, higher accuracy results at, for some instances, lower run-times. ILS-FDD with `3-opt` also compares well with the so far best performance results obtained by Merz and Freisleben with their genetic local search approach. In [19] they report average times for finding optimal solutions on instances `lin318,pcb442,att532,rat783` in 25, 35, 131, and 61 seconds, respectively on a 300MHz Pentium II CPU running Solaris (averages are taken over 30 runs). We obtained the same solution quality (except `att532` on which 4 of 100 runs did not yield the optimum) at slightly larger run-times on the two larger instances. Yet, we strongly believe that by using a faster `LK` implementation like Johnson's, Merz's or the code by Applegate, Cook, Bixby and Chvatal, the performance of ILS-FDD can be further strongly increased. Also, note that the only other approach which matches the results of ILS-FDD on the strongly clustered instances `fl1577` and `fl3795` is the ITP algorithm. Yet, our approach has the advantage of being an easy implementable extension of an `I3opt` algorithm. When using `LK` local search, we obtained very high accuracy results for all the instances tested, comparable or better than the results reported in other researches. Still, the computation times of the `LK` could be significantly improved by using one of the previously mentioned more fine-tuned implementations. All in all, our computational results show that ILS and, in particular, the proposed ILS extensions are highly competitive with more complex algorithms for the TSP.

6 Conclusions

In this paper, we presented a novel method for the systematic empirical analysis of local search algorithms based on run-time distributions. While this methodology has been previously successfully applied to combinatorial decision problems, such as SAT [9], here, we use it for investigating and improving the behavior of Iterated Local Search algorithms for the well-known Traveling Salesman Problem.

Our RTD-based analysis revealed that ILS algorithms for the TSP often show stagnation behavior, such that from some point in the search process, relatively little progress is made with respect to finding higher quality solutions. This phenomenon is observed not only for relatively simple ILS algorithms based on `2-opt` and `3-opt` local search, but also for the more complex `ILK` algorithm, one of the best performing approximate algorithms for large symmetric TSP instances, as well as for ILS algorithms for the asymmetric TSP. Based on this observation, we proposed improved variants of ILS algorithms for the TSP. In an experimental analysis we verified that our strategies to overcome stagnation behavior are effective and that the modified algorithms show a significantly improved performance.

These results demonstrate, that the RTD-based approach provides a good basis for analyzing and improving the performance of Iterated Local Search Algorithms in particular, and other meta-heuristics in general. We expect that by systematically applying this methodology to other algorithms and problem domains, further new insights into the behavior of meta-heuristics for hard combinatorial problems can be obtained which will facilitate the development of new, improved algorithms and their successful application.

Acknowledgements

We would like to thank Olivier Martin for making available his `ILK` implementation and for helpful comments on this work. This work was in part supported by a Marie Curie Fellowship awarded to Thomas Stützle (CEC-TMR Contract No. ERB4001GT973400) and a Postdoctoral Fellowship awarded by the University of British Columbia to Holger H. Hoos.

Bibliography

[1] R.K. Ahuja and J.B. Orlin. Use of Representative Operation Counts in Computational Testing of Algorithms. *INFORMS Journal on Computing*, 8(3):318–330, 1996.

[2] E.B. Baum. Iterated Descent: A Better Algorithm for Local Search in Combinatorial Optimization Problems. Manuscript, 1986.

[3] J.L. Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.

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

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA, 1979.

[6] P. Hansen and N. Mladenović. Variable Neighbourhood Search for the p-Median. Technical Report Les Cahiers du GERAD G-97-39, GERAD and École des Hautes Études Commerciales, 1997.

[7] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, Boston, 1999.

[8] H.H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms — Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 238–245. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[9] H.H. Hoos and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 1999. To appear.

[10] D.S. Johnson. Local Optimization and the Travelling Salesman Problem. In *Proc. 17th Colloquium on Automata, Languages, and Programming*, volume 443 of *LNCS*, pages 446–461. Springer Verlag, 1990.

[11] D.S. Johnson and L.A. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[12] K. Katayama and H. Narihisa. Iterated Local Search Approach using Genetic Transformation to the Traveling Salesman Problem. In *Proceedings of GECCO'99*, pages 321–328, 1999.

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Travelling Salesman Problem.* John Wiley & Sons, 1985.

[14] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[15] O. Martin and S.W. Otto. Partitoning of Unstructured Meshes for Load Balancing. *Concurrency: Practice and Experience*, 7:303–314, 1995.

[16] O. Martin and S.W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.

[17] O. Martin, S.W. Otto, and E.W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991.

[18] P. Merz and B. Freisleben. Genetic Local Search for the TSP: New Results. In *Proceedings of ICEC'97*, pages 159–164. IEEE Press, 1997.

[19] P. Merz and B. Freisleben. Fitness Landscapes and Memetic Algorithm Design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization.* McGraw-Hill, 1999.

[20] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24:1097–1100, 1997.

[21] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber. Combinatorial Optimization by Iterative Partial Transcription. *Physical Review E*, 59(4), 1999.

[22] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *LNCS*. Springer Verlag, 1994.

[23] A. Rohe. Parallele Heuristiken für sehr große Traveling Salesman Probleme. Master's thesis, Fachbereich Mathematik, Universität Bonn, Bonn, Germany, 1997.

[24] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. PhD thesis, FB Informatik, TU Darmstadt, 1998.

[25] T. Walters. Repair and Brood Selection in the Traveling Salesman Problem. In A.E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Proc. of Parallel Problem Solving from Nature – PPSN V*, volume 1498 of *LNCS*, pages 813–822. Springer Verlag, 1998.

[26] C. Young, D.S. Johnson, D.R. Karger, and M.D. Smith. Near-optimal Intraprocedural Branch Alignment. In *Proceedings 1997 Symp. on Programming Languages, Design, and Implementation*, pages 183–193, 1997.