J Stat Phys (2007) 129: 623–648
DOI 10.1007/s10955-007-9382-1

On the Neighborhood Structure of the Traveling Salesman Problem Generated by Local Search Moves

Günther Stattenberger · Markus Dankesreiter · Florian Baumgartner · Johannes J. Schneider

Received: 31 January 2007 / Accepted: 22 July 2007 / Published online: 2 September 2007

Abstract When trying to find approximate solutions for the Traveling Salesman Problem with heuristic optimization algorithms, small moves called Lin-k-Opts are often used. In our paper, we provide exact formulas for the numbers of possible tours into which a randomly chosen tour can be changed with a Lin-k-Opt. Furthermore, we compare the quality of the results to which the various moves lead.

Keywords Traveling Salesman Problem · TSP · Neighborhood

1 Introduction: The Traveling Salesman Problem

Due to the simplicity of its formulation and the complexity of its exact solution, the traveling salesman problem (TSP) has been studied for a very long time [1] and has drawn great attention from various fields, such as applied mathematics, computational physics, and operations research. The traveling salesman faces the problem to find the shortest closed tour through a given set of nodes, touching each of the N nodes exactly once and returning to the starting node at the end [1, 2]. Hereby the salesman knows the distances $d(i, j)$ between all

G. Stattenberger
Computer Science, Fernfachhochschule Schweiz, Überlandstr. 12, Postfach 689, 3900 Brig, Switzerland
e-mail: gstattenberger@fernfachhochschule.ch

M. Dankesreiter
Independent IT Consultant, Ed.-Mühlbauer-Weg 14, 93051 Regensburg, Germany
e-mail: Markus.Dankesreiter@web.de

F. Baumgartner
Swisscom Innovations, Swisscom, Ostermundigenstr. 93, 3006 Berne, Switzerland
e-mail: florian.baumgartner@swisscom.com

J.J. Schneider (✉)
Institute of Physics, Johannes Gutenberg University of Mainz, Staudinger Weg 7, 55099 Mainz, Germany
e-mail: schneidj@uni-mainz.de

pairs (i, j) of nodes, which are usually given as some constant non-negative values, either in units of length or of time. The costs $\mathcal{H}$ of a configuration σ are therefore given as the sum of the distances of the used edges. If denoting a configuration as a permutation σ of the numbers $\{1, \ldots, N\}$, the costs can be written as

$$\mathcal{H}(\sigma) = d(\sigma(N), \sigma(1)) + \sum_{i=1}^{N-1} d(\sigma(i), \sigma(i+1)). \tag{1}$$

A TSP instance is called symmetric if $d(i, j) = d(j, i)$ for all pairs (i, j) of nodes. For a symmetric TSP, the costs for going through the tour in a clockwise direction are the same as going through in an anticlockwise direction. Thus, these two tours are to be considered as identical.

As the time for determining the optimum solution of a proposed TSP instance grows exponentially with the system size, the number N of nodes, a large variety of heuristics has been developed in order to solve this problem approximately. Besides the application of several different construction heuristics [3], which were either specifically designed for the TSP or altered in order to enable their application to the TSP, the TSP has been tackled with various general-purpose improvement heuristics, like Simulated Annealing [4] and related algorithms such as Threshold Accepting [5, 6], the Great Deluge algorithm [7–9], algorithms based on the Tsallis statistics [10], Simulated and Parallel Tempering (methods described in [11–15]), and Search Space Smoothing [16–18]. Furthermore Genetic algorithms [19–21], Tabu Search [22] and Scatter Search [22, 23], and even Ant Colony Optimization [24], Particle Swarm Optimization [25–27], and other biologically motivated algorithms [28] have been applied to the TSP. The quality of these algorithms is compared by creating solutions for benchmark instances, one of which is shown in Fig. 1.

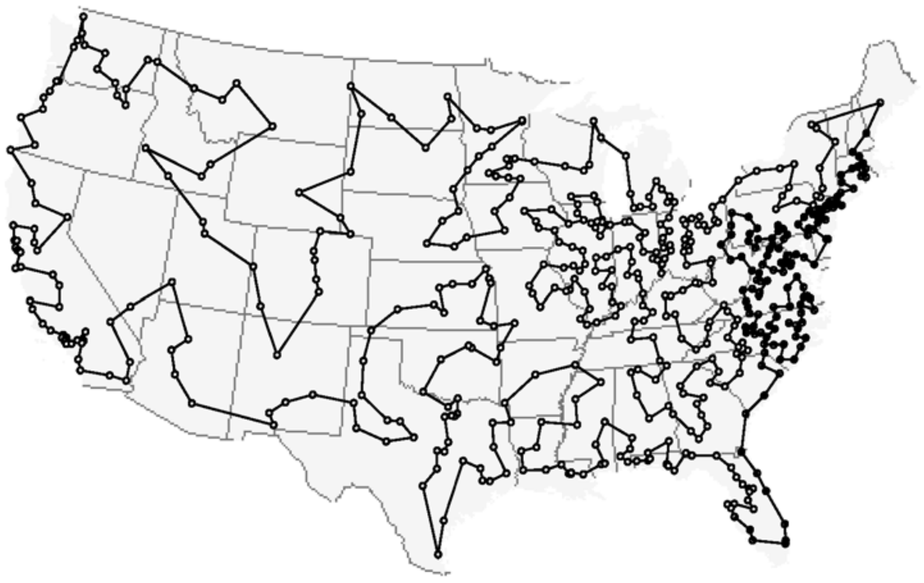


Fig. 1 The optimum solution of the ATT532 benchmark TSP instance containing the 532 AT&T switch locations in the United States of America: this benchmark instance is part of Reinelt's benchmark library TSPLIB95 [29]. The optimum solution was created with the Searching for Backbones algorithm [30, 31], making use of the Lin-2-Opt and the Lin-3-Opt, which are shown in Figs. 3 and 4

Most of these improvement heuristics apply a series of so-called small moves to a current configuration or a set of configurations. In this context, the move being small means that it does not change a configuration very much, such that usually the cost of the new tentative configuration which is to be accepted or rejected according to the acceptance criterion of the improvement heuristics does not differ very much from the cost of the current configuration. This method of using only small moves is called the Local Search approach, as the small moves lead to tentative new configurations, which are close to the previous configuration according to some metric like the Hamming distance for the TSP: the Hamming distance between two tours is given by the number of different edges.

These moves create a neighborhood structure between the various configurations. Two configurations σ and τ are considered to be neighbors of each other if there is a move m with $m(\sigma) = \tau$, i.e., if there is a move transferring configuration σ to configuration τ. When working with small moves, usually the neighborhood structure is symmetric, i.e., if there is a move $m : \sigma \mapsto \tau$, then there is also the inverse move $m^{-1} : \tau \mapsto \sigma$. Commonly, the description of a small move contains many possibilities for random decisions where and how to change something. According to which of the many possibilities is chosen the move applied to configuration σ ends up at one configuration τ_i or at an other configuration τ_j. The set $\mathcal{N}(\sigma) = \{\tau_1, \tau_2, \ldots, \tau_M\}$ is comprised of all these configurations and is called the neighborhood of configuration σ. The cardinal number $M = |\mathcal{N}(\sigma)|$ is called the neighborhood size of configuration σ. Usually, this neighborhood size of a move is identical for all configurations. In this paper, we aim at deriving exact values for the neighborhood sizes of various small moves.

2 The Smallest Moves

2.1 The Exchange

One move which does not change a configuration very much is the Exchange (EXC), which is sometimes also called Swap and which is shown in Fig. 2. The Exchange exchanges two randomly selected nodes in the tour. Thus, from a proposed configuration, $N \times (N-1)/2$ other configurations can be reached, such that the neighborhood of a configuration generated by this move has a size of order $\mathcal{O}(N^2)$.

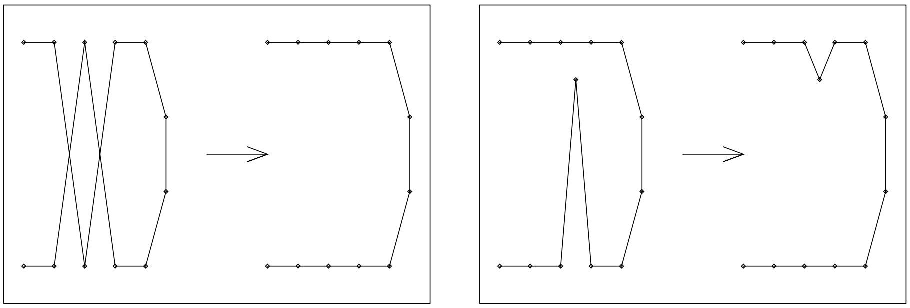


Fig. 2 The Exchange (*left*) and the Node Insertion Move (*right*)

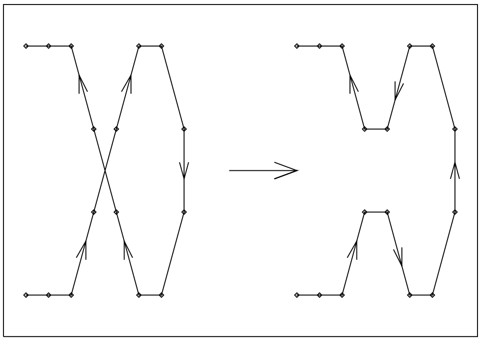


Fig. 3 The Lin-2-Opt

2.2 The Node Insertion Move

Another small move is the Node Insertion Move (NIM), which is also called Jump. It is also shown in Fig. 2. The Node Insertion Move randomly selects a node and an edge. It removes the randomly chosen node from its original position and places it between the end points of the randomly selected edge, which is cut for this purpose. The neighborhood size generated by this move is $N \times (N-2)$ and thus also of order $\mathcal{O}(N^2)$.

2.3 The Lin-2-Opt

Lin introduced a further small move, which is called Lin-2-Opt (L2O) [32, 33]: as shown in Fig. 3, it cuts two edges of the tour, turns the direction of one of the two partial sequences around, and reconnects these two sequences in a new way. For symmetric TSP instances, only the two removed edges and the two added edges have to be considered when calculating the cost difference created by this move. For these symmetric TSPs, it plays no role which of the two partial sequences is turned around when performing the move, due to the identical cost function value for moving through clockwisely or anticlockwisely. In the symmetric case, on which we will concentrate throughout this paper, the move creates a neighborhood of size $N \times (N-3)/2$ and thus of order $\mathcal{O}(N^2)$. Please note that a move cutting two edges after neighboring nodes does not lead to a new configuration, such that the neighborhood size is not $N \times (N-1)/2$, a false value which is sometimes found in the literature.

The Lin-2-Opt turned out to provide better results for the symmetric Traveling Salesman Problem than the Exchange. The reason for this quality difference was explained analytically by Stadler and Schnabl. In their paper [34], they basically found out that the results are the better the less edges are cut: the Lin-2-Opt cuts only two edges whereas the Exchange cuts four. But they also reported results that the Lin-3-Opt cutting three edges leads to an even better quality of the solutions than the Lin-2-Opt, what contradicted their results at first sight, but they explained this finding with the larger neighborhood size of the Lin-3-Opt.

3 The Lin-3-Opt

The next larger move to the Lin-2-Opt is the Lin-3-Opt (L3O): the Lin-3-Opt removes three edges of the tour and reconnects the three partial sequences to some new closed tour. In contrast to the smallest moves for which there is only one possibility to create a new tour, there are four possibilities in the case of the symmetric TSP how to create a new tour with three

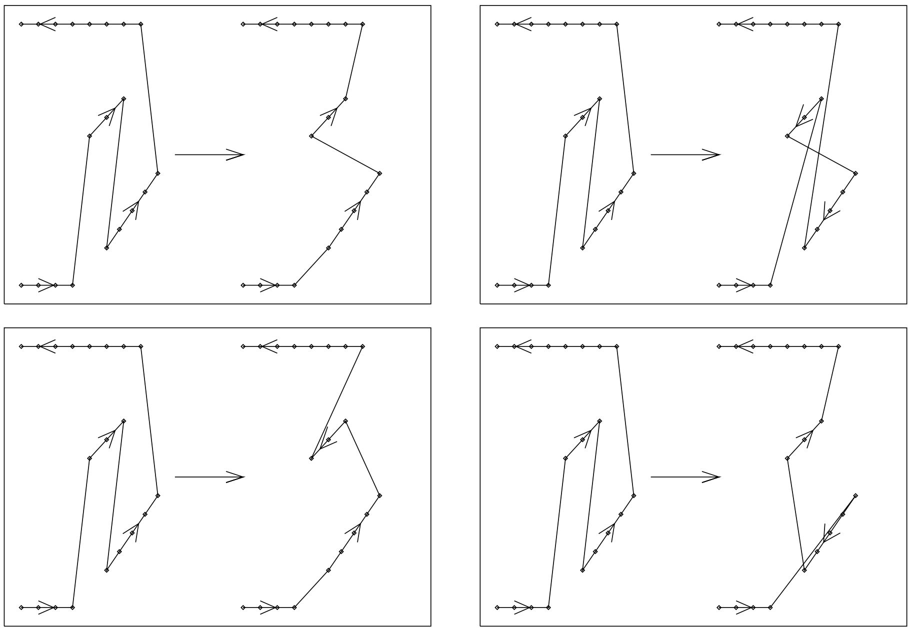


Fig. 4 The four possibilities how to change a tour with a Lin-3-Opt

new edges with the Lin-3-Opt if each of the partial sequences contains at least two nodes. These four possibilities are shown in Fig. 4. Please note that we count only the number of "true" possibilities here, i.e., only those cases in which the tour contains three edges which were not part formerly in the tour, as otherwise the move would e.g. only be a Lin-2-Opt. If one of the partial sequences contains only one node and the other two at least two nodes each, then only one possibility for a "true" Lin-3-Opt remains. If even two of the three partial sequences do only contain one node, then there is no possibility left to reconnect the three sequences without reusing at least one of the edges which was cut. Analogously, there is one possibility for the Lin-2-Opt, if both partial sequences contain at least two nodes each, otherwise there is no possibility.

If looking closely at the four variants of the Lin-3-Opt in Fig. 4, one finds that the resulting configurations could also be generated by a sequence of Lin-2-Opts: for the upper left variant in Fig. 4, three Lin-2-Opts would be needed, whereas only two Lin-2-Opts would be sufficient for the other three variants. Thus, one might ask whether the Lin-3-Opt is necessary as a move as a few Lin-2-Opts could do the same job. However, due to the acceptance criteria of the improvement heuristics, it might be that at least one of the Lin-2-Opts would be rejected whereas the combined Lin-3-Opt move could be accepted. Thus, it is often advantageous also to implement these next-higher-order moves in order to overcome the barriers in the energy landscape of the small moves.

Now the question arises how large the neighborhood size of a Lin-3-Opt is. Of course, it has to be of order $\mathcal{O}(N^3)$, as three edges to be removed are randomly selected out of N edges. However, for the calculation of the exact number of possibilities one has to distinguish between the case in which all partial sequences contain at least two nodes each and the case in which exactly one partial sequence contains only one node.

Please note that the Node Insertion Move, which was introduced earlier, is the special case of the Lin-3-Opt in which one of the partial sequences only contains one node. But in the special case that one of the two next nearest edges to the randomly chosen node is selected, the Node Insertion Move corresponds to a Lin-2-Opt. As the number of cut edges of the Node Insertion Move is 3, such that this move is basically a Lin-3-Opt, but as the neighborhood size of this move is of order $\mathcal{O}(N^2)$, this move is also sometimes called Lin-2.5-Opt.

4 Higher-Order Moves

One can go on to even higher-order Lin-k-Opts: the Lin-4-Opt cuts four edges of the tour and reconnects the four created partial sequences in a new way. If every partial sequence contains at least two nodes, there are 25 possibilities for a true Lin-4-Opt to reconnect the partial sequences to a closed feasible tour. The neighborhood size of this move is of order $\mathcal{O}(N^4)$.

The Exchange, which was also introduced earlier, is usually a special case of a Lin-4-Opt. Only if the two nodes which are to be exchanged are direct neighbors of each other or if there is exactly one node between them, then the move is equivalent to a Lin-2-Opt.

One can increase the number k of deleted edges further and further. However, by doing so, one gradually loses the advantage of the Local Search approach, in which, due to the similarity of the configurations, their cost values do not differ much. In the extreme, the Lin-N-Opt would lead to a randomly chosen new configuration, the cost value of which is not related to the cost value of the previous configuration at all. Moving away from the Local Search approach, the probability for getting an acceptable new configuration among the many more neighboring configurations with cost values in a much larger interval strongly decreases due to the finite available computing time. When using the Local Search approach, it turns out that using the smallest possible move only is only optimal in the case of very short computing times. With increasing computing time, a well chosen combination of the smallest moves and their next larger variants becomes optimal. Here the optimization run has more time to search through a larger neighborhood. The next larger moves enable the system to overcome barriers in the energy landscape formed by the small moves only. Of course, one can extend this approach and also include moves with the next larger k and spend even more computing time.

However, for some difficult optimization problems, an approach based on small moves and their next larger variants is not sufficient [35]. There indeed large moves have to be used. A successful approach here are the Ruin & Recreate moves, which destroy a configuration to some extent and rebuild it according to a given rule set. They work in a different way than the small moves, which completely randomly select a way to change the configuration. In contrast, the Ruin & Recreate moves contain constructive elements in order to result in good configurations. Also for problems like the TSP, for which small moves basically work, well designed Ruin & Recreate moves are superior to the small moves [35]. However, the development of excellent Ruin & Recreate moves is rather difficult, it is indeed an optimization problem itself, whereas the application of the Local Search approach, which simply intends to "change the configuration a little bit", is rather straightforward and also usually quite successful in producing good solutions, such that it is mostly used.

Sometimes one needs to know the exact size of the neighborhood generated by the implemented moves for relating it to the available computing time or the available amount of computer memory. This is especially the case for optimization algorithms like Tabu Search, which e.g. forbid to perform a move which is inverse to moves which were performed only a short while ago. In this case, such forbidden moves are stored in a tabu list. In the extreme case, the number of entries in this tabu list, the so-called tabu list size, is equal to the neighborhood size. For cases like these, one needs to know exactly whether the proposed problem can still be treated with that optimization algorithm on a computer with a rather limited amount of memory. Furthermore, there are some gradient techniques like Steepest Descent, which search in the neighborhood of the current configuration for the best neighboring configuration which is then accepted as new configuration. The calculation time of such algorithms increases linearly with the neighborhood size. Naturally, one is aware of the neighborhood size of a Lin-k-Opt being of order $\mathcal{O}(N^k)$, but as just mentioned, one at least additionally needs to know the leading prefactor. The aim of this paper is to provide exact numbers for the neighborhood size. For deriving the neighborhood size of the Lin-3-Opt and of even larger Lin-k-Opts, we will start with the determination of the number of possibilities for reconnecting partial sequences to a complete tour with a true Lin-k-Opt in Sect. 5. There we will find laws how many possibilities exist depending on the number of partial sequences containing only one node and their spatial neighborhood relation to each other. Having this distinction at hand, we will calculate the corresponding numbers of possibilities for cutting the tour in Sect. 6.

5 Number of Possibilities for Reconnecting the Tour with a Lin-k-Opt

5.1 Special Case

5.1.1 Number of Overall Possibilities

For the calculation of the number of possibilities for reconnecting the tour, we want to start out with the special case that each partial sequence contains at least two nodes. A Lin-k-Opt cuts the tour into k partial sequences and reconnects them using k new edges which were not part of the previous configuration. Thus, the neighborhood of an arbitrary configuration σ contains all configurations which differ in k edges from σ. However, as a first step we consider the overall number of possibilities to reconnect these k partial sequences to a closed tour containing all nodes, not caring about whether the edges to be inserted were used in the previous configuration or not. This overall number can be obtained when imagining the following scenario: one randomly selects one of the partial sequences and fixes its direction. (This has to be done for the symmetric TSP for which a tour and its mirror tour are degenerate.) This first partial sequence serves as a starting sequence for the new tour to be constructed. Then one randomly selects one out of the $k-1$ remaining partial sequences and adds it to the already existing partial tour. There are two possible ways of adding it, one for each direction of the partial sequence. Thus, one gets a new system with only $k-1$ partial sequences. The number of possibilities to construct a new feasible tour is thus given as

$$P(k) = 2(k-1)P(k-1). \tag{2}$$

This recursive formula can be easily desolved to

$$P(k) = 2^{k-1}(k-1)!. \tag{3}$$

Table 1 Number of true Lin-k-Opts if the TSP is symmetric and if every partial sequence contains at least 2 nodes

k	$T(k)$
0	1
1	0
2	1
3	4
4	25
5	208
6	2121

5.1.2 Number of True Lin-k-Opts

But this overall set of possibilities contains many variants in which old edges which were cut are reused in the new configuration, such that the move is not a true Lin-k-Opt. Thus, in order to get the number of true Lin-k-Opts, those variants have to be subtracted from the overall number. As there are $\binom{k}{i}$ possibilities to choose i old edges for the new tour if there were overall k deleted edges, the number of true Lin-k-Opts is given by the recursive formula

$$T(k) = 2^{k-1}(k-1)! - \sum_{i=0}^{k-1} \binom{k}{i} T(i) \quad \text{with } T(0) = 1. \tag{4}$$

The starting point of this recursion is $T(0) = 1$, as there is one possibility for the Lin-0-Opt, the identity move, in which no edge is changed. Table 1 gives an overview of the numbers of true Lin-k-Opts for small k, in the case that each partial sequence contains at least two nodes and that the TSP is symmetric. We find that there is one Lin-0-Opt, the identity move, no Lin-1-Opt, as by cutting only one edge no new tour can be formed, one Lin-2-Opt, four Lin-3-Opts, and so on. In the case of an asymmetric TSP, each number here has to be multiplied by a factor of 2.

5.2 General Case

5.2.1 Number of Overall Possibilities

Again we start considering all types of moves which delete k edges, thus cut the tour into k partial sequences, and then reconnect these sequences to one closed tour by adding k edges, not caring about whether these edges were part of the previous configuration.

In the general case, not every partial sequence contains at least two nodes. There might be sequences containing only one node which are surrounded by two sequences containing more than one node in the old tour. Furthermore, there might be tuples of neighboring sequences containing only one node each which are surrounded by two sequences containing more than one node, and so on.

Let α_0 be the number of the partial sequences containing more than one node, α_1 be the number of sequences containing only one node and surrounded by two sequences containing more than one node in the old tour, α_2 be the number of tuples of one-node-sequences surrounded by two sequences containing more than one node in the old tour, α_3 be the number of triples of one-node-sequences surrounded by two sequences containing more than one node in the old tour, and so on. We will see that P and T are no longer functions of k only, but depend on the entries of the vector $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots)$.

In the following, the assumption shall hold that not every edge of the tour is cut, such that $k < N$ and $\alpha_0 > 0$. Thus, one can always choose a partial sequence consisting of two or more nodes as a starting point for the creation of a new tour and fix its direction, as we only consider the symmetric TSP here.

Starting with this fixed partial sequence, a new feasible tour containing all nodes can be constructed by iteratively selecting an other partial sequence and adding it to the end of the growing partial tour. For the overall number of possibilities for construcing a new tour, it plays no role whether one-node-sequences were side by side in the old tour or not. Thus, the number of possibilities $P(\vec{\alpha})$ is simply given as

$$P(\vec{\alpha}) = P(\alpha_0, k - \alpha_0, 0, \dots, 0). \tag{5}$$

There are two possible ways for adding a partial sequence containing at least two nodes, but only one possibility for adding a sequence with only one node. Thus, analogously to the result above we get the result

$$P(\vec{\alpha}) = 2^{\alpha_0 - 1}(k-1)!. \tag{6}$$

For the asymmetric TSP, this number has to be multiplied with 2.

5.2.2 Number of True Lin-k-Opts

When calculating the number of true Lin-k-Opts, i.e., the number of moves which delete k edges and add k new edges which are not identical with the deleted edges, we need to consider the spatial arrangement of the one-node-sequences in the old tour. We have to distinguish between single one-node-sequences, tuples, triples, quadruples, and so on, i.e., we have to consider the i-tuples for each i separately. Contrarily, we have no problems with the spatial arrangement of partial sequences with at least two nodes. In order to get the number of true Lin-k-Opts, we want to use a trick by artificially blowing up partial sequences with only one node to sequences with two nodes. Let us first consider here that there are not only partial sequences with at least two nodes but also isolated partial sequences with only one node. (We thus first leave out the tuples, triples, … of single-node-sequences in our considerations, but it does not matter here whether there are any such structures.) We extend one one-node sequence to two nodes by doubling the node. Thus, one gets $T(\alpha_0 + 1, \alpha_1 - 1, \dots)$ possibilities for performing a true Lin-k-Opt instead of $T(\alpha_0, \alpha_1, \dots)$ possibilities. By changing the direction of this blown-up sequence, one can connect it—in contrast to before as it consisted of only one node—to those nodes of the neighboring parts to which it was connected before. There are two possibilities to connect it this way to one of the two neighboring partial sequences and one possibility to connect it this way to both of them. But these cases are forbidden, such that we have to subtract the number of these possibilities in which they get connected and we achieve the recursive formula

$$\begin{aligned} T(\vec{\alpha}) = \frac{1}{2}(&T(\alpha_0 + 1, \alpha_1 - 1, \alpha_2, \dots) - 2T(\alpha_0, \alpha_1 - 1, \alpha_2, \dots) \\ &- T(\alpha_0 - 1, \alpha_1 - 1, \alpha_2, \dots)). \end{aligned} \tag{7}$$

Please note that the resulting number has to be divided by 2, as a partial sequence with only one node cannot be inserted in two different directions.

Analogously, one can derive a formula if there are tuples of neighboring sequences with only one node each. Here one expands one of the two partial sequences to two nodes, such that there is one tuple less, but one isolated one-node-sequence more and one longer sequence more. Analogously to above, the false possibilities must be subtracted and the result divided by 2, such that we get the formula

$$\begin{aligned}T(\vec{\alpha}) = \frac{1}{2}(&T(\alpha_0+1,\alpha_1+1,\alpha_2-1,\alpha_3,\ldots) - T(\alpha_0,\alpha_1+1,\alpha_2-1,\alpha_3,\ldots)\\ &- T(\alpha_0+1,\alpha_1,\alpha_2-1,\alpha_3,\ldots) - T(\alpha_0,\alpha_1,\alpha_2-1,\alpha_3,\ldots)).\end{aligned} \tag{8}$$

For all longer groups of single-node-sequences, like triples and quadruples, there is one common approach. Here it is appropriate to blow up a single-node-sequence at the frontier, such that the following recursive formula is achieved:

$$\begin{aligned}T(\vec{\alpha}) = \frac{1}{2}(&T(\alpha_0+1,\alpha_1,\ldots,\alpha_{i-1}+1,\alpha_i-1,\alpha_{i+1},\ldots)\\ &- T(\alpha_0,\alpha_1,\ldots,\alpha_{i-1}+1,\alpha_i-1,\alpha_{i+1},\ldots)\\ &- T(\alpha_0+1,\alpha_1,\ldots,\alpha_{i-2}+1,\alpha_{i-1},\alpha_i-1,\alpha_{i+1},\ldots)\\ &- T(\alpha_0,\alpha_1,\ldots,\alpha_{i-2}+1,\alpha_{i-1},\alpha_i-1,\alpha_{i+1},\ldots)).\end{aligned} \tag{9}$$

Generally, one should proceed with the recursion in the following way: first, those nonzero α_i should vanish for which i is maximal. This approach should be iterated with decreasing i until one ends up for a formula for tours with partial sequences consisting of at least two nodes each for which we can use the formula for the special case.

6 Number of Possibilities for Cutting the Tour with a Lin-k-Opt

6.1 Special Case

After having determined the number of possibilities to reconnect partial sequences to a closed tour with a true Lin-k-Opt, i.e., with a move leading to a configuration with k new edges, we still have to determine the number of possibilities for cutting the tour in order to create these partial sequences. We again start out with the special case in which every partial sequence to be created shall contain at least two nodes. For small values of k, the number of these possibilities can be found empirically: if considering the Lin-2-Opt, one sees at first sight that there are N possibilities for performing the first cut. Let us denote the position of the node after which the tour is cut as i. Then in order to get two partial sequences with at least two nodes each, the tour must not be cut after one of the tour positions $i-1$, i, or $i+1$, such that there are $N-3$ possibilities for performing the second cut. As the possibilities for performing the first cut and the possibilities for the second cut have to be multiplied and as the first and the second cut can be exchanged, one ends up at the overall number $N \times (N-3)/2!$ of possibilites for cutting the tour with a Lin-2-Opt.

Now let us derive the number of possibilities for the Lin-3-Opt, for which a graphical illustration like in Fig. 5 is very useful: there are N possibilities for performing the first cut, let us without restriction assume that this cut is performed after tour position 1, as shown in the figure. Then there are various possibilities for performing the second cut:

- Cutting the tour after tour position 2 is impossible, as then a partial sequence with only one node would be created. Thus, the first possibility for performing the second cut is after tour position 3. Then the question arises how many possibilities for performing the

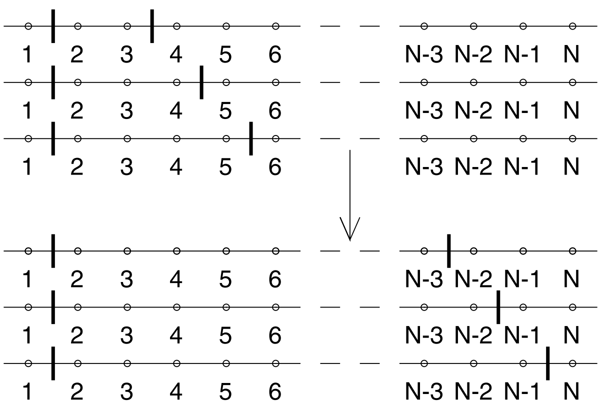


Fig. 5 Illustration for the derivation of the number of possibilities for cutting the tour into three partial sequences containing at least two nodes each: the first cut is fixed here between the tour positions 1 and 2. For each position of the second cut, which is subsequently shifted through the whole tour, the number of possibilities for the third cut, which is not shown here, has to be determined. Please note that there is of course an edge between the first and the last node in the tour and that it is not allowed to perform the second cut after the tour positions 2 and N as a partial sequence containing only one node would be created in these two cases

third cut exist for this scenario that the first cut was performed after tour position 1 and the second after 3. Of course, the tour must not be cut after the tour positions 1 and 3 again, furthermore, the tour must not be cut after the tour positions 2, 4, and N, as then a partial sequence consisting of only one node would be created. Thus, $N-5$ possibilities for performing the third cut remain.

- The next possibility for performing the second cut is after tour position 4. Again we have to ask how many possibilities there are for performing the third cut: the tour must not be cut after the tour positions 1 and 4 again, furthermore, it must not be cut after the tour positions 2, 3, 5, and N. The third cut is only allowed after the tour positions $6, 7, 8, \ldots, N-1$, which makes $N-6$ possibilities for the third cut.
- Next there is the possibility to perform the second cut after tour position 5. In this scenario, the third cut is allowed only after the tour positions $7, 8, 9, \ldots, N-1$ and also after the tour position 3, such that we again find $N-6$ possibilities.
- Shifting the second cut to even larger tour position numbers, the number of possibilities for performing the third cut between the first and the second cut increases in the same way as the number of possibilities for performing the third cut after the second cut decreases. Thus, if the second cut is performed after tour position $N-3$, then we have again $N-6$ possibilities for performing the third cut as it is only forbidden to perform the third cut after the tour positions $1, 2, N-4, N-3, N-2$ and N.
- The case that the second cut is performed after tour position $N-2$ is analogous to the case in which it was performed after tour position 4, such that we have also here $N-6$ possibilities for performing the third cut.
- The last possibility for the second cut is to perform it after tour position $N-1$. The third cut cannot be performed after the tour positions $1, 2, N-2, N-1$, and N, such that $N-5$ possibilities remain.

Summarizing, we have N possibilities for performing the first cut and $N-3$ possibilities for performing the second cut. Two of these $N-3$ possibilities lead to scenarios in which we have $N-5$ possibilities for performing the third cut, whereas the remaining $N-5$

Table 2 Number of possibilities for cutting the tour of a traveling salesman if each partial sequence shall contain at least two nodes: k denotes the number of cuts of the Lin-k-Opt, $C(k)$ the number of possibilities

k	$C(k)$
2	$N \times (N-3)/2$
3	$N \times (N-4)(N-5)/3!$
4	$N \times (N-5)(N-6)(N-7)/4!$
5	$N \times (N-6)(N-7)(N-8)(N-9)/5!$

possibilities of the second cut lead to $N-6$ possibilities for the third cut. Thus, we have overall $N \times (2 \times (N-5) + (N-5) \times (N-6)) = N \times (N-4) \times (N-5)$ possibilities for performing the cuts in this order. As an exchange in the temporal order of the first, the second, and the third cut leads to the same partial sequences, we have to divide this overall number by 3!. Therefore, we get $N(N-4)(N-5)/3!$ possibilities in the case of a Lin-3-Opt.

This analysis can be performed in the same way for even larger values of k. Of course, finding all cases becomes more difficult but the scheme sketched above can be used unchanged. By empirical going through all possibilities, we found the formulas which are given in Table 2. From this result, we deduce a general formula for the possibilities $C(k)$ for the Lin-k-Opt:

$$C(k) = N \times \prod_{i=1}^{k-1}(N-k-i) \times \frac{1}{k!} = \frac{N}{N-k} \times \binom{N-k}{k}. \tag{10}$$

Thus, we find here that the neighborhood created by a Lin-k-Opt is of order $\mathcal{O}(N^k)$.

6.2 General Case

In the general case, an arbitrary Lin-k-Opt can also lead to partial sequences containing only one node. Here we have to distinguish between various types of cuts: the cuts introduced by a Lin-k-Opt can be isolated, i.e., they are between two sequences with more than one node each. Then they can lead to isolated nodes which are between two partial sequences with more than one node each, and so on. Let us view this from the point of view of the cuts of the tour. All in all, a Lin-k-Opt generally leads to k cuts in the tour. Let us denote an i-type multicut (with $1 \le i \le k$) at position j (with $1 \le j \le N$) as the scenario that the tour is cut at i successive positions after the node with the tour position number j. Thus, the tour is cut by an i-type multicut successively between pairs of nodes with the tour position numbers $(j, j+1), (j+1, j+2), \dots, (j+i-1, j+i)$.

Furthermore, let β_i be the number of i-type multicuts: β_1 is the number of isolated cuts. β_2 is the number of 2-type multicuts by which the tour is cut after two successive tour positions such that a partial sequence containing only one node is created, surrounded by two sequences containing more than one node. Thus, β_2 is also the number of isolated nodes surrounded by longer sequences and is thus identical with α_1. Analogously, 3-type multicuts lead to tuples of nodes which are surrounded by partial sequences with more than one node, thus, β_3 is the number α_2 of these tuples. Analogously, 4-type multicuts lead to triples of sequences containing only one node each, and so on. Generally, we have for all $i \ge 1$ that

$$\alpha_i = \beta_{i+1} \tag{11}$$

of the last section, but for $i = 0$ the situation is different: each i-type multicut produces a further sequence consisting of at least two nodes, such that we have

$$\alpha_0 = \sum_{i=1}^{k} \beta_i. \tag{12}$$

Please note that α_0 is both the number of these longer partial sequences and the number of all i-type multicuts. The overall number k of cuts can be expressed as

$$k = \sum_{i=1}^{k} i\beta_i. \tag{13}$$

In this general case, the number C of ways of cutting the tour also depends not only on k, but on the entries of the vector $\vec{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$. As Table 3 shows, the order

Table 3 Number of possibilities for cutting the tour of a traveling salesman: k denotes the overall number of cuts performed by the Lin-k-Opt, β_1, β_2, β_3, and β_4 denote the number of 1-, 2-, 3-, and 4-type multicuts as defined in the text. Only β_i-values for $i \le 4$ are considered here in our examples. For the Lin-6-Opt, only some special cases are considered. The number C of possibilities depends on all these numbers. All these formulas for $C(\vec{\beta})$ were found by hand

k	β_1	β_2	β_3	β_4	C
2	2	0	0	0	$N \times (N-3)/2$
	0	1	0	0	N
3	3	0	0	0	$N \times (N-4)(N-5)/3!$
	1	1	0	0	$N \times (N-4)$
	0	0	1	0	N
4	4	0	0	0	$N \times (N-5)(N-6)(N-7)/4!$
	2	1	0	0	$N \times (N-5)(N-6)/2$
	0	2	0	0	$N \times (N-5)/2$
	1	0	1	0	$N \times (N-5)$
	0	0	0	1	N
5	5	0	0	0	$N \times (N-6)(N-7)(N-8)(N-9)/5!$
	3	1	0	0	$N \times (N-6)(N-7)(N-8)/3!$
	1	2	0	0	$N \times (N-6)(N-7)/2$
	2	0	1	0	$N \times (N-6)(N-7)/2$
	0	1	1	0	$N \times (N-6)$
	1	0	0	1	$N \times (N-6)$
6	2	2	0	0	$N \times (N-7)(N-8)(N-9)/2/2$
	0	3	0	0	$N \times (N-7)(N-8)/3!$
	3	0	1	0	$N \times (N-7)(N-8)(N-9)/3!$
	1	1	1	0	$N \times (N-7)(N-8)$
	0	0	2	0	$N \times (N-7)/2$
	2	0	0	1	$N \times (N-7)(N-8)/2$
	0	1	0	1	$N \times (N-7)$

of the neighborhood size is now given as $\mathcal{O}(N^{\alpha_0})$. From these examples in the table, we empirically derive the formula

$$C(\vec{\beta}) = N \times \prod_{i=1}^{\sum_{j=1}^{k} \beta_j - 1} (N - k - i) \times \frac{1}{\prod_{i=1}^{k} \beta_i!} \tag{14}$$

for the number of possibilities for cutting a tour with a Lin-k-Opt, leading to β_i many i-type multicuts. Please note that if the upper index of a product is smaller than the lower index, then this so-called empty product is 1. This formula can be rewritten to

$$C(\vec{\beta}) = \frac{N}{N-k} \times \binom{N-k}{\alpha_0} \times \frac{\alpha_0!}{\prod_{i=1}^{k} \beta_i!} \tag{15}$$

making use of α_0 being the sum of all β_i. Please note that (10) for the special case with each sequence containing at least two nodes is a special case of (15).

The numbers for cutting a tour in partial sequences in this section were first found by hand for the examples given in Tables 2 and 3, then the general formulas were intuitively deduced from these. After that the correctness of these formulas was checked by computer programs up to $k = 20$ both for the special case and for all variations of the general case.

7 The Neighborhood Size of a Lin-k-Opt

Now let us summarize and combine the results we have achieved so far: first of all, the multicut structure to which a specific Lin-k-Opt leads has to be examined, as one has to distinguish between isolated cuts, which divide two partial sequences with at least two nodes, two cuts just behind each other, such that a partial sequence with only one node is created, which is in between two partial sequences with more than one node, then three cuts just behind each other, such that a tuple of partial sequences with only one node each is created, and so on. The number of possibilities for cutting a tour according to these structures of multicuts is given in (15). From the numbers of multicut structures, one has then to derive the numbers of partial sequences, which are given in (11) and (12). Then one has to use the recursive formulas (7), (8), and (9) in order to simplify the dependency of the number of reconnections to only one parameter. After that one has to use (4) for getting the number of reconnections for a true Lin-k-Opt with k new edges. Finally, one has to sum up the products of the numbers of possible cuttings and of the numbers of possible reconnections in order to get the overall number of neighboring configurations which can be reached via the move.

8 Quality of the Results Achieved with Various Moves

In the second part of our investigations, we want to ask which quality of solutions can be achieved with the various moves. Here we concentrate on a comparison of the quality which can be achieved by the "power" of the moves only, thus, we do not use some elaborate underlying heuristics like those mentioned in the introduction, but use the simplest algorithm, which is often called Greedy algorithm: starting out from a randomly chosen configuration σ_0, a series of moves is applied to the system. Each move chooses randomly a way how to change the current configuration σ_i into a new configuration σ_{i+1}. A move is only accepted by the Greedy algorithm if it does not lead to a deterioration, i.e., if the length $\mathcal{H}(\sigma_{i+1})$ of the

Table 4 Results for the Greedy algorithm using small moves (Exchange, Node Insertion Move, and Lin-2-Opt) and the variants of the Lin-3-Opt: for each instance (BEER127, LIN318, and PCB442), 100 optimization runs were performed, starting with a random configuration and performing a specific number (given in the text) of the corresponding move

Instance	Move	Minimum	Maximum	Mean value	±	Error
BEER127	EXC	159705.087	208409.703	182672.403	±	977.4
	NIM	132898.075	161648.200	146418.726	±	596.4
	L2O	121178.315	138126.861	129964.407	±	325.5
	all small	119331.431	134236.701	126605.995	±	305.2
	L3O1	119218.371	131235.436	125118.494	±	254.3
	L3O2	120935.053	133226.981	125888.805	±	260.4
	L3O3	118589.344	127980.167	121981.992	±	187.4
	L3O4	118899.537	127588.470	121928.533	±	187.4
	L3Oall	118629.043	127881.507	121396.175	±	194.1
LIN318	EXC	90116.6104	127763.120	110660.750	±	628.5
	NIM	59992.7967	78436.1451	68407.9569	±	376.2
	L2O	45115.9243	49971.4471	47276.4710	±	95.0
	all small	43488.2914	47875.0510	45743.4573	±	78.9
	L3O1	43883.1117	46992.0997	45210.0271	±	63.9
	L3O2	44209.8503	47368.0175	45466.4980	±	66.1
	L3O3	42863.7951	44822.3800	43632.8142	±	39.5
	L3O4	42626.6703	45173.6630	43534.8133	±	44.4
	L3Oall	42401.9352	44514.0820	43518.6735	±	40.7
PCB442	EXC	112322.878	142974.558	129290.471	±	708.4
	NIM	64129.9159	79939.0300	70764.4069	±	296.7
	L2O	54682.4205	59026.1119	56614.6608	±	82.3
	all small	52589.4816	56840.8241	54660.7005	±	76.3
	L3O1	53547.9002	56348.0251	54850.7759	±	60.7
	L3O2	52975.6709	57838.1568	54847.6013	±	85.7
	L3O3	51658.9173	54163.2202	52689.2427	±	47.9
	L3O4	51627.2806	53519.6159	52545.4050	±	41.0
	L3Oall	51480.2480	53892.2666	52493.4278	±	44.5

tentative new tour is as long as or is shorter than the length $\mathcal{H}(\sigma_i)$ of the current tour. After the acceptance of the move $\sigma_i \to \sigma_{i+1}$, the system tries to move from the configuration σ_{i+1} to a new configuration. In case of rejection of the move $\sigma_i \to \sigma_{i+1}$, one sets $\sigma_{i+1} = \sigma_i$ and proceeds as above.

Tables 4 and 5 show the minimum, maximum, and average quality of solutions which can be achieved with the various moves for five different TSP benchmark instances which were taken from Reinelt's TSPLIB95 [29] and which vary in size between $N = 127$ and $N = 1379$. In order to be quite sure to end up in a local minimum, we used $50 \times N^2$ move trials if using either the Exchange or the Node Insertion Move or the Lin-2-Opt. If comparing the results achieved with these small moves, we find that the Lin-2-Opt leads to the best results and the Exchange leads to the worst results, a result which is in accordance with the theoretical derivations in [34]. Now one can also mix these moves in the way that a

Table 5 Results as in Table 4 for the instances ATT532 and NRW1379

Instance	Move	Minimum	Maximum	Mean value	±	Error
ATT532	EXC	69294	91946	80037.86	±	449.7
	NIM	37691	47862	42460.43	±	212.1
	L2O	29960	32026	30867.73	±	42.2
	all small	29069	30530	29689.64	±	31.1
	L3O1	28716	30639	29732.22	±	33.7
	L3O2	29229	30817	30031.61	±	30.7
	L3O3	28275	29369	28692.05	±	22.8
	L3O4	28076	29159	28673.47	±	21.4
	L3Oall	28281	29102	28718.27	±	19.6
NRW1379	EXC	164670.848	196316.798	181017.337	±	607.0
	NIM	89353.7705	105344.742	94486.5234	±	294.3
	L2O	62862.6385	64887.1349	64046.8071	±	42.9
	all small	60334.7758	62208.5391	61154.0388	±	39.4
	L3O1	64624.4812	66712.2867	65611.2830	±	35.6
	L3O2	63267.3838	64788.3074	64008.2578	±	37.1
	L3O3	58730.9852	59768.7048	59199.8516	±	25.4
	L3O4	58727.9973	59656.5893	59173.3947	±	23.5
	L3Oall	58650.2000	60077.9761	59307.8550	±	26.1

general move routine calls each of these three moves with equal probability. The next lines in Tables 4 and 5 show that the results, which were taken after $150 \times N^2$ move trials, are better for this mixture than for any of the three moves alone. This is of course due to the larger neighborhood size of this mixture.

Then we provide the results for the four variants of the Lin-3-Opt, for which the results were taken after $10 \times N^3$ move trials (a few test runs froze after $1-2 \times N^3$ move trials), and for a mixture (here we call each of the four variants with equal probability), for which the results were taken after $20 \times N^3$ move trials. If we denote the tour position numbers after which the tour is cut by the Lin-3-Opt as i, j, and k with $i < j < k$ and their subsequent tour position numbers as i_+, j_+, and k_+, we can write the cut tour as follows:

$$\ldots \quad \sigma(i)\big|\sigma(i_+) \quad \ldots \quad \sigma(j)\big|\sigma(j_+) \quad \ldots \quad \sigma(k)\big|\sigma(k_+) \quad \ldots .$$

Please note that the tour is of course closed, such that the partial sequence starting at $\sigma(k_+)$ ends with $\sigma(i)$. If $j \neq i_+$, $k \neq j_+$, and $i \neq k_+$, there are four possibilities to reconnect the three partial sequences with a true Lin-3-Opt:

$$\begin{array}{llc|ccc|ccc|cl}
\text{L3O1:} & \ldots & \sigma(i) & \sigma(j_+) & \ldots & \sigma(k) & \sigma(i_+) & \ldots & \sigma(j) & \sigma(k_+) & \ldots, \\
\text{L3O2:} & \ldots & \sigma(i) & \sigma(j) & \ldots & \sigma(i_+) & \sigma(k) & \ldots & \sigma(j_+) & \sigma(k_+) & \ldots, \\
\text{L3O3:} & \ldots & \sigma(i) & \sigma(j_+) & \ldots & \sigma(k) & \sigma(j) & \ldots & \sigma(i_+) & \sigma(k_+) & \ldots, \\
\text{L3O4:} & \ldots & \sigma(i) & \sigma(k) & \ldots & \sigma(j_+) & \sigma(i_+) & \ldots & \sigma(j) & \sigma(k_+) & \ldots .
\end{array}$$

Tables 4 and 5 show that there are some differences in the qualities of the results achieved with the four variants of the Lin-3-Opt and that the mixture provides the best results for the three smaller instances, whereas two variants are on average better than the mixture for

Table 6 Results for the Greedy algorithm using two types of Lin-4-Opts: for each instance, 100 optimization runs were performed, starting with a random configuration and performing N^4 times one variant of the Lin-4-Opt with the Greedy acceptance criterion. As the computing time is rather large, only small instances are considered

Instance	Move	Minimum	Maximum	Mean value	$\pm$	Error
BEER127	L4O1	123767.073	134986.024	129588.275	$\pm$	244.2
	L4O2	121407.255	131300.030	125622.863	$\pm$	214.7
LIN318	L4O1	45184.3246	48569.5839	46740.9896	$\pm$	72.2
	L4O2	44539.2015	46932.1063	45451.7793	$\pm$	48.3
PCB442	L4O1	55673.1534	58855.2662	57381.3213	$\pm$	71.3
	L4O2	53793.3942	56843.4938	55209.2331	$\pm$	64.0

the instances with $N = 532$ and $N = 1379$. Comparing these results with the results for the small moves, we find that the four variants of the Lin-3-Opt lead to a much better quality of the results than the small moves, which is in accordance with the results in [36]. As Stadler and Schnabl already mentioned in [34], this better quality of the results has to be expected because of the much larger neighborhood size of the Lin-3-Opts.

Now one can ask why not to proceed and to move on to even larger moves. We implemented two of the 25 different variants of the Lin-4-Opt. If denoting the tour position numbers after which the tour is cut as i, j, k, and l and their subsequent numbers as i_+, j_+, k_+, and l_+, then the cut tour can be written as follows:

$$\ldots \quad \sigma(i)\big|\sigma(i_+) \quad \ldots \quad \sigma(j)\big|\sigma(j_+) \quad \ldots \quad \sigma(k)\big|\sigma(k_+) \quad \ldots \quad \sigma(l)\big|\sigma(l_+) \quad \ldots .$$

Then the move variants lead to the following new tours:

$$\begin{array}{lllllllllllll}
\text{L4O1:} & \ldots & \sigma(i)\big| & \sigma(k_+) & \ldots & \sigma(l)\big| & \sigma(j_+) & \ldots & \sigma(k)\big| & \sigma(i_+) & \ldots & \sigma(j)\big| \sigma(l_+) & \ldots, \\
\text{L4O2:} & \ldots & \sigma(i)\big| & \sigma(k_+) & \ldots & \sigma(l)\big| & \sigma(k) & \ldots & \sigma(j_+)\big| & \sigma(i_+) & \ldots & \sigma(j)\big| \sigma(l_+) & \ldots .
\end{array}$$

The variant L4O1 is sometimes called the two-bridge-move, as the two “bridges” $\sigma(i_+)\ldots\sigma(j)$ and $\sigma(k_+)\ldots\sigma(l)$ are exchanged. Table 6 shows the quality of the results achieved with these two variants of the Lin-4-Opt. We find that a further improvement cannot be found, the results for the better variant of the Lin-4-Opt are roughly of the same quality as the results for the worse variants of the Lin-3-Opt. Thus, leaving the Local Search approach even further leads to worse results, especially, if only a small amount of computing time is available, as the time to find a “good move” increases with increasing neighborhood size.

Of course, using only the Greedy algorithm, one fails in achieving the global optimum configurations for the TSP instances, which have a length of 118293.52... (BEER127 instance), 42042.535... (LIN318 instance), 50783.5475... (PCB442 instance), 27686 (ATT532 instance, for which the distances are calculated using the ATT metric according to TSPLIB95 [29], which we also use for this instance), and 56638 (NRW1379 instance, when the distances are rounded to integers), respectively. These optima can be achieved with the small moves and the variants of the Lin-3-Opt if using a better underlying heuristic (see e.g. [30, 31, 37]).

9 Results for Simulated Annealing

One such heuristic which provides much better results than the Greedy algorithm is Simulated Annealing [4]. In contrast to the Greedy algorithm, which forbids any deterioration and thus soon gets stuck in a high lying local minimum near the starting point of the optimization run, Simulated Annealing additionally accepts deteriorations with probability $\exp(-\Delta\mathcal{H}/T)$, with $\Delta\mathcal{H}$ being the energy difference between the current and the tentative new configuration and T being the temperature. The temperature, which is simply a control parameter here, governing the acceptance probability of moves leading to worse configurations, is gradually reduced during the optimization run: at high temperatures, most deteriorations are allowed. In the $T \to 0$-limit, Simulated Annealing approaches the same scenario as the Greedy algorithm, by not allowing deteriorations anymore. For the Traveling Salesman Problem, a cooling schedule in which the temperature is decreased as $T_{\text{new}} = \alpha \times T_{\text{old}}$ is preferable [38].

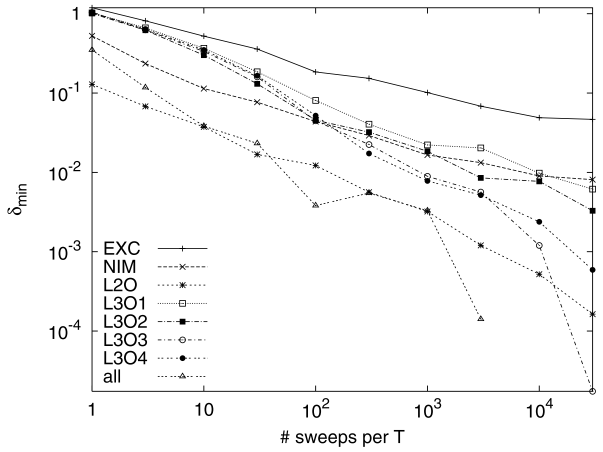
The most important question arising now is whether this order of the quality of the various moves, namely that the Lin-3-Opt is better than the Lin-2-Opt which is in turn better than the Node Insertion Move which is in turn better than the Exchange, is preserved if using a better underlying heuristic like Simulated Annealing. Here we present only results for the PCB442 instance, but the conclusions can be made generally. The optimization runs were performed in the same way as in [38], i.e., the PCB442 instance was cooled down from the initial temperature 10^4 to 0.1 with a cooling factor of 0.99. Finally, one Greedy step was performed, such that 1147 temperature steps were performed. Thus, the move routine was called 1147 (the number of temperature steps) times the number of sweeps per temperature times 442 (the number of TSP nodes). In order to have a good view at the quality differences between the various moves, here we consider the minimum and mean relative deviation of the achieved results to the optimum,

$$\delta_{\min} = \frac{\mathcal{H}_{\min} - \mathcal{H}_{\text{opt}}}{\mathcal{H}_{\text{opt}}} \quad \text{and} \quad \delta_{\text{mean}} = \frac{\langle\mathcal{H}\rangle - \mathcal{H}_{\text{opt}}}{\mathcal{H}_{\text{opt}}} \tag{16}$$

with $\mathcal{H}_{\text{opt}} = 50783.5475\ldots$ being the optimum tour length of the PCB442 instance. The average is taken over 100 optimization runs.

Figure 6 shows the results for various calculation times, depending on the move used. In the "all"-scenario, every move (Exchange, Node Insertion Move, Lin-2-Opt, and the four variants of the Lin-3-Opt) was called with equal probability. At first sight, we find that the results differ strongly for the various moves: the worst results are clearly achieved with the Exchange for all calculation times. For short calculation times, the Lin-2-Opt leads to the best results, followed by the "all moves"-scenario, and the Node Insertion Move. Thus, if having only a rather small amount of calculation time, it is the best to work with a good small move only and not to use larger moves. However, for some new problem, usually one does not know in advance which of the small moves lead to good results. Furthermore, in this time regime, which lasts only between milliseconds and seconds depending on the system size it is better first to use a construction heuristics and then—if there is still time available—to proceed with an afterburner in the Greedy mode.

For larger amounts of computing time, the average results for using the Lin-2-Opt only or working with all moves coincide. Three variants of the Lin-3-Opt provide better results than the Node Insertion Move. The Exchange still leads to the worst results. Looking at $\delta_{\min}$, we find that the "all moves"-scenario finally finds the optimum (these are the "missing points" in the curve, as the $\delta_{\min}$-axis is drawn in a logarithmic way) if the number of sweeps per temperature step is at least 10^4, whereas the Lin-2-Opt alone fails in finding the optimum.



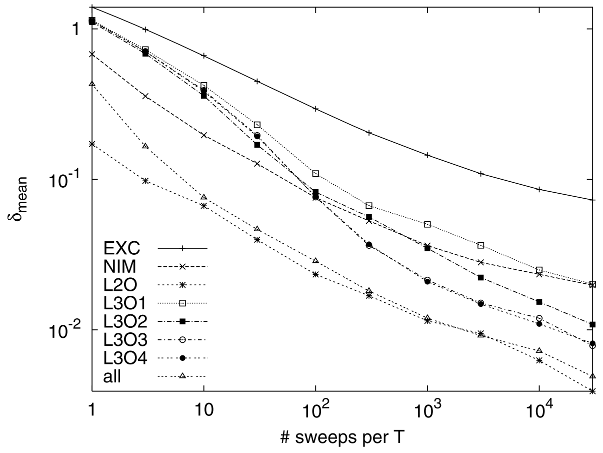


Fig. 6 Minimum and mean deviation δ (as defined in formula (16)) vs. calculation time if applying Simulated Annealing to the PCB442 instance

Summarizing, we want to state that when dealing with a new proposed problem it is the best to implement as many smallest order and next higher order moves as possible. The often heard sentence "you only need to implement one small move, as the algorithm will do it all." Is simply false or at least not generally true. Certainly, one might find a move like the Lin-2-Opt for a specific problem, which leads to rather the same quality of results as the ensemble of moves does, but then it has to be the right move and not e.g. the analogon to the Exchange. This additional implementation work pays off in the quality of the results, especially if more calculation time is invested.

As the order "Exchange worse than Node Insertion Move worse than Lin-2-Opt" is still preserved when using a better heuristic like Simulated Annealing, we now want to focus on these three smallest moves only and continue studying their behaviors in the "greedy" scenario.

10 Results for Steepest Descent

A variant of the Greedy algorithm is the Steepest Descent method, which is also sometimes called Greedy algorithm. The Steepest Descent algorithm, which usually also starts at a

Table 7 Results for the application of the Steepest Descent method to the BEER127, LIN318, PCB442, and ATT532 TSP instances, using either the Exchange or the Node Insertion Move or the Lin-2-Opt as move: the results for the number of steps until no further improvement can be found and for the final lengths are averaged over 100 optimization runs

Instance	Move	Steps	±	Error	Final length	±	Error
BEER127	EXC	156.01	±	17.5	188731.586	±	11677.0
	NIM	175.50	±	13.6	148452.222	±	6001.2
	L2O	129.71	±	5.2	126245.082	±	2753.0
LIN318	EXC	522.64	±	46.8	122639.534	±	7323.3
	NIM	602.39	±	32.7	71957.503	±	3674.7
	L2O	378.74	±	8.9	45387.867	±	696.5
PCB442	EXC	835.69	±	62.4	145677.130	±	6900.6
	NIM	880.25	±	42.1	73007.457	±	3466.2
	L2O	504.99	±	9.7	56168.044	±	873.8
ATT532	EXC	1108.78	±	78.0	89917.88	±	4850.5
	NIM	1135.17	±	49.1	46257.06	±	3293.4
	L2O	643.26	±	12.5	30186.47	±	356.1

randomly chosen configuration, performs iteratively a complete search of the whole neighborhood of the current configuration for the largest improvement. Then the best neighboring configuration is accepted as new solution. This scheme is iterated until no improvement is found anymore.

Table 7 shows the results for the Steepest Descent method: we find also here that using the Lin-2-Opt leads to better results than the usage of the Node Insertion Move which is in turn better than using the Exchange. When comparing the Greedy algorithm with Steepest Descent, one might think that the gradient technique Steepest Descent has to be superior to the Greedy algorithm, as Steepest Descent always chooses the largest improvement whereas the Greedy algorithm randomly selects improving moves. However, when also looking at Tables 4 and 5, we find that this superiority is only found in the case of using the Lin-2-Opt, whereas the usage of the Node Insertion Move or of the Exchange leads to worse results for the Steepest Descent method.

Furthermore, Table 7 shows the number of steps the Steepest Descent method needs until finding no improvement anymore. This number is equal to the number of accepted moves. We find that using the Lin-2-Opt is fastest in getting stuck in a local minimum. The number of steps needed when using the Node Insertion Move is only slightly smaller than the number of steps needed when using the Exchange which is slowest at finding a local minimum.

As we had to perform a complete search of the neighborhood of each configuration we visited anyway, when applying the Steepest Descent method, we also counted for each of these configurations the fraction f of moves which would either lead to an improvement or at least to a equally good configuration. The results of this counting are shown in a histogram with a logarithmic energy scale in Fig. 7. We find that the fraction f generally decreases with decreasing energy, i.e., the better a configuration already is, the less neighbors it has which are better than it. Furthermore, we generally find that f is largest for the Lin-2-Opt and smallest for the Exchange.

Please note that the values for this fraction might slightly change if using an other underlying heuristic: when using the Steepest Descent technique, the system moves downhill at

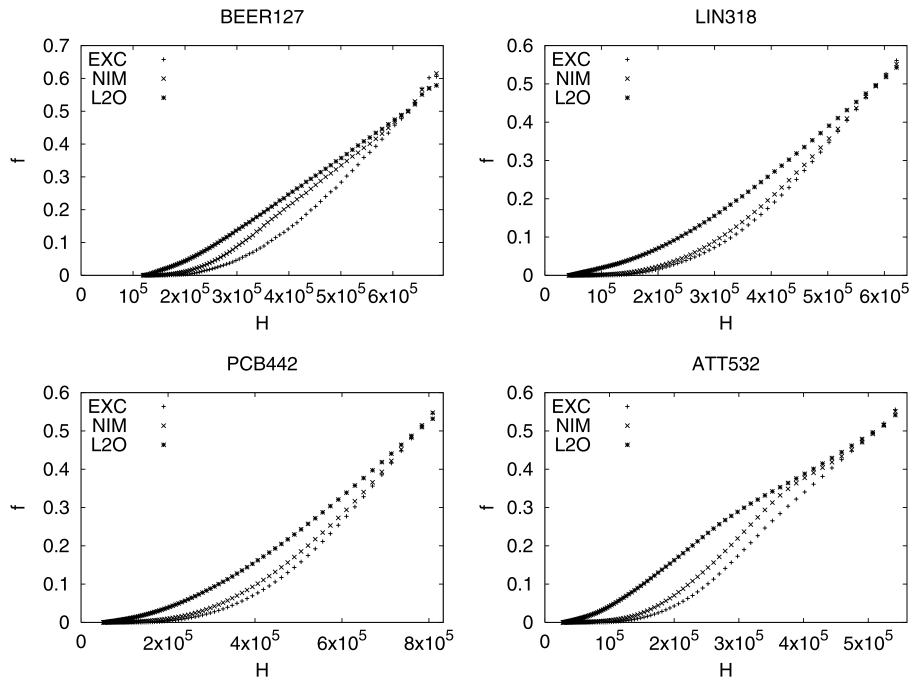


Fig. 7 Fraction f of moves leading to either better or equally good configurations vs. energy $\mathcal{H}$ of the current configuration, for the moves Exchange, Node Insertion Move, and Lin-2-Opt and for the TSP instances BEER127, LIN318, PCB442, and ATT532: the results are averaged over 100 optimization runs

the bottom of a sinking channel in the energy landscape. Contrarily, when using Simulated Annealing, the system is still able to climb over small barriers even at low temperatures. Therefore, one might get slightly larger values for f when using such a method.

11 Promising Moves

In the first part of this paper, we showed that even small moves like the Lin-2-Opt create a neighborhood structure containing e.g. $N(N-3)/2$ for each configuration. When moving on to even higher moves or when increasing the number N of nodes, the number of possibilities for tentative new configurations explodes. On the other hand, we found in the last section that the neighborhood of a rather good configuration contains only a very small fraction of configurations which are better. Therefore, the random choice of a neighboring configuration, which is usually performed when using the Greedy algorithm or more elaborate heuristics like Simulated Annealing seems to be a waste of computing time.

The question arises whether this large size of the neighborhood can be broken down to the improving moves only. Of course, these improving moves are not known a priori. But are there some promising candidates within this large set of moves for which one would expect that they might on average lead to better results than other candidates and that they lead to these results in a very short calculation time?

An obvious way of choosing such promising candidates would be to use the spatial neighborhood between the various nodes, an approach which was already introduced by Dueck and Scheuer for the Lin-2-Opt in [5]: as we aim at finding a roundtrip of minimum length for the traveling salesman, it would at first sight make sense to connect every node with two of its nearest neighbors in order to decrease the lengths of the corresponding edges. We have implemented this new approach for the three small moves as follows:

- "promising" Exchange:
  First a node a is randomly selected. Then a node b which is one of the nearest neighbors of a is randomly selected and which is not the successor of a in the roundtrip. Then the successor of a and the node b are exchanged.
- "promising" Node Insertion Move:
  First a node a is randomly selected after which the tour is to be cut. Then a node b which is one of the nearest neighbors of a is randomly selected under the condition that b is not successor of a in the roundtrip. Then the node b is moved between a and the successor of a.
- "promising" Lin-2-Opt:
  First a node a is randomly selected. Then a node b which is one of the nearest neighbors of a is randomly selected. If b is not the successor of a in the roundtrip, then the tour is cut after the nodes a and b and a Lin-2-Opt is performed, such that node b becomes the successor of node a.

Table 8 Results for the approach using promising move candidates (pEXC) with $\#NN = 5$ and $\#NN = 10$ nearest neighbors, respectively, in comparison to the original way of choosing move variants at random (oEXC), applied for the Exchange: for each instance (BEER127, LIN318, PCB442, ATT532, and NRW1379), 100 optimization runs were performed, starting with a random configuration and performing $N \times \#NN$ moves

Instance	Move	$\#NN$	Minimum	Maximum	Mean value	±	Error
BEER127	pEXC	5	186070.070	250479.537	214608.660	±	1199.5
	oEXC	5	214220.253	263034.849	241480.217	±	1027.7
	pEXC	10	169336.900	235755.252	199781.876	±	1151.1
	oEXC	10	194002.966	238887.679	215628.087	±	969.3
LIN318	pEXC	5	137673.010	186869.924	160594.884	±	964.1
	oEXC	5	159203.446	200404.119	182773.789	±	657.4
	pEXC	10	122251.982	160447.156	138385.482	±	732.9
	oEXC	10	142056.390	173897.354	158801.768	±	647.1
PCB442	pEXC	5	172120.501	223090.572	196386.635	±	1006.3
	oEXC	5	218330.288	256177.115	240679.810	±	670.1
	pEXC	10	145712.204	190090.969	168055.642	±	923.3
	oEXC	10	188413.932	222203.496	208099.606	±	724.7
ATT532	pEXC	5	113898	145278	127313.67	±	608.6
	oEXC	5	137502	167169	150940.03	±	533.6
	pEXC	10	91509	122720	106473.59	±	578.3
	oEXC	10	116365	141658	130462.07	±	489.8
NRW1379	pEXC	5	312918.168	353345.897	333376.791	±	985.2
	oEXC	5	411445.312	456756.333	428841.044	±	748.4
	pEXC	10	251660.750	300854.280	276771.296	±	1078.9
	oEXC	10	349942.916	390667.147	367629.841	±	750.6

Table 9 Results as in Table 8 but now for the promising (pNIM) and the original (oNIM) variants of the Node Insertion Move

Instance	Move	$\#NN$	Minimum	Maximum	Mean value	±	Error
BEER127	pNIM	5	156079.584	215166.942	183205.812	±	1049.8
	oNIM	5	194884.573	240826.533	218480.947	±	841.4
	pNIM	10	146375.370	178893.499	162198.791	±	760.7
	oNIM	10	172824.125	212001.153	189321.381	±	731.8
LIN318	pNIM	5	103943.282	146871.561	125404.861	±	813.3
	oNIM	5	144013.542	169922.712	156346.312	±	603.8
	pNIM	10	81913.999	113822.999	96240.876	±	653.7
	oNIM	10	114789.504	140841.654	128246.464	±	546.7
PCB442	pNIM	5	100750.549	133770.936	116188.918	±	767.4
	oNIM	5	185965.158	226675.510	204825.220	±	700.1
	pNIM	10	73148.149	103147.869	89593.809	±	571.3
	oNIM	10	150615.509	180378.511	166641.501	±	554.1
ATT532	pNIM	5	70736	104804	89080.75	±	660.7
	oNIM	5	115655	138473	128274.39	±	457.5
	pNIM	10	52875	72020	62480.95	±	451.2
	oNIM	10	93085	114698	103226.44	±	394.5
NRW1379	pNIM	5	170492.325	203206.626	189134.147	±	757.3
	oNIM	5	351414.807	383144.570	367676.291	±	654.9
	pNIM	10	122357.469	153649.690	134900.715	±	709.6
	oNIM	10	284494.833	314429.450	298792.043	±	548.4

Tables 8, 9, and 10 show results for these "promising" variants of the moves in comparison to their original counterparts for two different numbers of considered nearest neighbors. Please note that for the original moves, the variable $\#NN$ only governs the amount of calculation time and that here very short computing times were used such that the optimization runs were hardly able to get into a local minimum. When comparing the results in Tables 8 and 9 with corresponding results in Tables 4 and 5 we find that due to the small amount of calculation time, neither the original nor the promising approach is able to produce results getting close to the energy values of the local minima which were achieved after a very large amount of calculation time, when having a look at the results for the Node Insertion Move and for the Exchange. However, the promising approach generally produces better results than the original approach. Doubling both the number of nearest neighbors and the calculation time for the promising approach leads to an improvement of the results. For the Exchange, we additionally find that the original approach performing 10 sweeps performs nearly as well as the promising approach performing only 5 sweeps, such that the promising approach basically saves 50% of the calculation time. This relation holds also true for small instances if the Node Insertion Move is used.

When looking at Table 10, we find two main results: first of all, the promising approach also leads in the case of the Lin-2-Opt to a large improvement of the results when compared to the original approach with the same amount of calculation time. Moreover, when comparing these results to those in Tables 4 and 5, we find that the promising approach leads to rather the same quality of the results within a small amount of calculation time, although

Table 10 Results as in Tables 8 and 9 but now for the promising (pL2O) and the original (oL2O) variants of the Lin-2-Opt

Instance	Move	$\#NN$	Minimum	Maximum	Mean value	±	Error
BEER127	pL2O	5	120914.336	147409.536	131137.023	±	493.5
	oL2O	5	158695.131	193975.734	174433.866	±	744.5
	pL2O	10	122993.783	139760.144	130364.863	±	366.5
	oL2O	10	135777.828	161549.605	148319.706	±	482.1
LIN318	pL2O	5	47800.331	62395.991	55047.699	±	333.4
	oL2O	5	95557.044	111293.035	104063.800	±	320.3
	pL2O	10	45295.842	52366.095	48424.877	±	170.5
	oL2O	10	72462.665	82961.264	77439.372	±	223.4
PCB442	pL2O	5	54502.263	67853.612	61183.729	±	253.3
	oL2O	5	129591.100	147885.638	137121.227	±	318.5
	pL2O	10	54376.765	59878.951	56724.305	±	109.0
	oL2O	10	98065.559	107799.185	102485.733	±	218.1
ATT532	pL2O	5	31689	42885	36357.26	±	207.7
	oL2O	5	74412	86027	81582.21	±	201.6
	pL2O	10	30053	34407	31543.01	±	96.3
	oL2O	10	54915	62790	59756.21	±	139.2
NRW1379	pL2O	5	66347.921	81274.952	71906.597	±	248.7
	oL2O	5	231210.754	250376.658	239907.919	±	371.3
	pL2O	10	61924.761	65395.623	63362.254	±	65.3
	oL2O	10	167950.389	181688.022	174756.115	±	265.5

we formerly invested a large amount in order to end up in locally minimum configurations. This result is again in accordance with the prediction in [34] that the Lin-2-Opt is the fastest move, i.e., that it can transverse the configuration space much faster than the Exchange and the Node Insertion Move.

Thus, one might want to conclude that one should generally only use this promising approach as it saves a lot of calculation time. However, when doing so one might fail to reach the globally optimum configuration as it might be necessary to solve the system locally in a bad way within an overall optimum configuration and as these promising moves alone might in such a case not be able to lead to the global optimum.

12 Conclusion

For getting an approximate solution of an instance of the Traveling Salesman Problem, mostly an improvement heuristic is used, which applies a sequence of move trials which are either accepted or rejected according to the acceptance criterion of the underlying heuristic. For the Traveling Salesman Problem, mostly small moves are used which do not change the configuration very much. Among these moves, the Lin-2-Opt, which cuts two edges of the tour and turns around a part of the tour, has been proved to provide superior results. Thus, this Lin-2-Opt and its higher-order variants, the Lin-k-Opts, which cut k edges of the tour and reconnect the created partial sequences to a new feasible solution by adding k new edges, and their properties have drawn great attention.

In this paper, we have provided formulas for the exact calculation of the number of configurations which can be reached from an arbitrarily chosen tour via these Lin-k-Opts. A specific Lin-k-Opt leads to a specific structure of multicuts, i.e., there are isolated cuts, which divide two partial sequences with at least two nodes, then there are two cuts just behind each other, such that a partial sequence with only one node is created, which is in between two partial sequences with more than one node, then there are three cuts just behind each other, such that a tuple of partial sequences with only one node each is created, and so on. We have derived both the number of possibilities for cutting a tour according to these structures of multicuts as well as the number of possibilities for reconnecting the partial sequences to a closed roundtrip, such that the overall neighborhood size of the move can be calculated.

In the second part of our investigations, we have compared the results achieved with these moves using the simple Greedy algorithm which rejects all moves leading to deteriorations. We have found that the Lin-2-Opt is superior to the other small moves and that the Lin-3-Opt provides even better results than the Lin-2-Opt whereas two variants of the Lin-4-Opt are worse than the Lin-3-Opt when using the Greedy algorithm. However, when applying Simulated Annealing, we find that the Lin-2-Opt produces the best results for short time scales whereas only a mixture of all small moves and of the Lin-3-Opt was able to find the global optimum for long time scales.

Then we compared the results for the Greedy algorithm with results achieved for the Steepest Descent gradient method. We found that accepting a better neighboring configuration which is randomly chosen leads to better results than choosing the best neighboring configuration if the Exchange or the Node Insertion Move is used, whereas a contrary result can be found for the Lin-2-Opt. Furthermore, we discovered that the better a configuration is, the less neighbors it has which are better than it.

Finally, we noticed that a breaking-down of the neighborhood size by only applying promising candidates of the moves leads to a large speedup and an improvement of the results for short computing times, compared to the scenario that the moves are chosen entirely at random. However, it depends on the proposed problem instance whether this “promising approach” is sufficient to reach the global optimum, even when using a better underlying heuristic.

When dealing with other optimization problems, one does not know in advance which small move leads to good results. Thus, we strongly recommend to implement all small moves and their next higher order counterparts. But moving even further away from the Local Search approach by adding even higher order moves usually does not lead to further improvements, at least as long as the tentative new configuration shall be selected entirely at random. Please note that approaches like Ruin & Recreate [35, 38], which change the configuration to a large extent, are only enabled to lead to good results, as the tentative new configuration is constructed according to a proposed rule set, ensuring that the move leads to a good configuration. But for such approaches, it is often rather difficult to find a good way for changing configurations to a large extent while ensuring that (nearly) all good configurations, including the global optimum, can be reached, whereas it is much easier to develop and to implement moves according to the Local Search paradigm.

Acknowledgements J.J.S. wants to thank Erich P. Stoll (University of Zurich) for the picture of the United States of America, which serves as a background in Fig. 1.

References

1. Der Handlungsreisende—wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein—von einem alten Commis-Voyageur (1832)
2. Lawler, E.L., Lenstra, J.K., Rinnoy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem. Wiley, New York (1985)
3. Reinelt, G.: The Traveling Salesman. Springer, Berlin (1994)
4. Kirkpatrick, S., Gelatt, C.D. Jr., Vecchi, M.P.: Science **220**, 671 (1983)
5. Dueck, G., Scheuer, T.: J. Comput. Phys. **90**, 161 (1990)
6. Moscato, P., Fontanari, J.F.: Phys. Lett. A **146**, 204 (1990)
7. Dueck, G.: J. Comput. Phys. **104**, 86 (1993)
8. Dueck, G., Scheuer, T., Wallmeier, H.-M.: Spektrum der Wissenschaft **1993**(3), 42 (1993)
9. Dueck, G.: Das Sintflutprinzip—Ein Mathematik-Roman. Springer, Heidelberg (2004)
10. Penna, T.J.P.: Phys. Rev. E **51**, R1–R3 (1995)
11. Marinari, E., Parisi, G.: Europhys. Lett. **19**, 451 (1992)
12. Kerler, W., Rehberg, P.: Phys. Rev. E **50**, 4220 (1994)
13. Hukushima, K., Nemoto, K.: J. Phys. Soc. Jpn. **65**, 1604 (1996)
14. Hukushima, K., Takayama, H., Yoshino, H.: J. Phys. Soc. Jpn. **67**, 12 (1998)
15. Coluzzi, B., Parisi, G.: J. Phys. A **31**, 4349 (1998)
16. Gu, J., Huang, X.: IEEE Trans. Syst. Man Cybern. **24**, 728 (1994)
17. Schneider, J., Dankesreiter, M., Fettes, W., Morgenstern, I., Schmid, M., Singer, J.M.: Physica A **243**, 77 (1997)
18. Coy, S.P., Golden, B.L., Wasil, E.A.: Eur. J. Oper. Res. **124**, 15 (2000)
19. Schöneburg, E., Heinzmann, F., Feddersen, S.: Genetische Algorithmen und Evolutionsstrategien. Addison-Wesley, Bonn (1994)
20. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
21. Holland, J.: SIAM J. Comput. **2**, 88 (1973)
22. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Norwell (1998)
23. Rego, C., Alidaee, B.: Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search. Kluwer Academic (2005)
24. Colorni, A., Dorigo, M., Maniezzo, V.: In: Proceedings of ECAL91—European Conference on Artificial Life, Paris, vol. 134 (1991)
25. Kennedy, J.: In: IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana (1997)
26. Kennedy, J., Eberhart, R.: In: Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, p. 1942 (1995)
27. Kennedy, J., Eberhart, R., Shi, Y.: Swarm Intelligence. Morgan Kaufmann, San Mateo (2001)
28. Stillinger, F.H., Weber, T.A.: J. Stat. Phys. **52**, 1429 (1988)
29. http://www.informatik.uni-heidelberg.de/groups/comopt/software/TSPLIB95
30. Schneider, J., Froschhammer, C., Morgenstern, I., Husslein, T., Singer, J.M.: Comput. Phys. Comm. **96**, 173 (1996)
31. Schneider, J.: Future Gener. Comput. Syst. **19**, 121 (2003)
32. Lin, S.: Bell Syst. Tech. J. **44**, 2245 (1965)
33. Lin, S., Kernighan, B.W.: Oper. Res. **21**, 498 (1973)
34. Stadler, P.F., Schnabl, W.: Phys. Lett. A **161**, 337 (1992)
35. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: J. Comput. Phys. **159**, 139 (1999)
36. Kirkpatrick, S., Toulouse, G.: J. Phys. **46**, 1277 (1985)
37. Schneider, J., Morgenstern, I., Singer, J.M.: Phys. Rev. E **58**, 5085 (1998)
38. Schneider, J.J., Kirkpatrick, S.: Stochastic Optimization. Springer, Berlin (2006)