Local Search for the Traveling Salesman Problem: A Comparative Study

Yuezhong Wu*, Thomas Weise*†, and Raymond Chiong‡
*Joint USTC-Birmingham Research Institute in Intelligent Computation and Its Applications (UBRI),
School of Computer Science and Technology, University of Science and Technology of China;
Hefei, Anhui, China, 230027. Emails: yuezhong@mail.ustc.edu.cn, tweise@ustc.edu.cn
†Corresponding Author.
‡School of Design, Communication and Information Technology,
Faculty of Science and Information Technology, The University of Newcastle;
Callaghan, NSW 2308, Australia. Email: Raymond.Chiong@newcastle.edu.au

***Abstract*—The Traveling Salesman Problem (TSP) is one of the most well-studied combinatorial optimization problems. Best heuristics for solving the TSP known today are Lin-Kernighan (LK) local search methods. Recently, Multi-Neighborhood Search (MNS) has been proposed and was demonstrated to outperform Variable Neighborhood Search based methods on the TSP. While LK performs a variable k-opt based search operation, MNS is able to carry out multiple 2-, 3-, or 4-opt moves at once, which are discovered by a highly efficient scan of the current solution. Apart from LK and MNS, many other modern heuristics for TSPs can be found in the relevant literature. However, existing studies rarely use robust statistics for the heuristic algorithms in comparison, let alone investigate their progress over time. This leads to flawed comparisons of simple end-of-run statistics and inappropriate or even incorrect conclusions. In this paper, we thoroughly compare LK and MNS as well as their hybrid versions with Evolutionary Algorithms (EAs) and Population-based Ant Colony Optimization (PACO). This work, to the best of our knowledge, is the first statistically sound comparison of the two efficient heuristics as well as their hybrids with EAs and PACO over time based on a large-scale experimental study. We not only show that hybrid PACO-MNS and PACO-LK are both very efficient, but also find that the full runtime behavior comparison provides deeper and clearer insights whereas a focus of final results could indeed have led to a deceitful conclusion.**

I. Introduction

The Traveling Salesman Problem (TSP) [1–3] is maybe the most important $\mathcal{NP}$-hard problem, both in terms of practical applications as well as being a test bed for novel optimization approaches. Given n cities, a salesman departs from a start city, visits each city exactly once, and then returns back to the start city. The task is to find the order in which the salesman should visit all of the cities while traveling the minimal overall distance. In other words, given a cost matrix $D = (D_{i,j})$, where $D_{i,j}$ is the distance of going from city i to city j ($i, j \in 1 \ldots n$), the goal is to find a permutation t of the integers from 1 to n minimizing the sum $D_{t[1],t[2]} + D_{t[2],t[3]} + \ldots + D_{t[n],t[1]}$. The focus of this study is the symmetric TSP, where $D_{i,j} = D_{j,i}$ holds.

Many algorithms for solving TSPs have been introduced. These include Local Search (LS) algorithms [4], Evolutionary Algorithms (EAs) [5–7], Ant Colony Optimization (ACO) [8–10], Branch and Bound (BB) [11] and hybrid algorithms [12]. Today, the best known LS methods for the TSP are the Lin-Kernighan (LK) heuristic [4] and its derivatives. In [13], an alternative approach, Multi-Neighborhood Search (MNS), was introduced and found to be a more efficient LS algorithm for the TSP than Variable Neighborhood Search [14] or Hill Climbing and better than pure EAs and Population-based ACO (PACO) [15]. PACO algorithms hybridized with MNS were shown to outperform all of the other tested algorithms, but no LK approach was benchmarked in that study (see [13] for details). In this work, we make the following contributions:

1) We provide an in-depth and statistically sound comparison of LK and MNS and show under which circumstances which of them performs better;
2) We introduce hybrid versions of both algorithms with EAs and PACO;
3) We conduct a large-scale experimental study and apply advanced runtime-behavior based statistics providing much more information about the performance of these algorithms than simple end-result comparisons. This study gathers more precise information, includes more experiments, and provides more statistically sound evaluations than any other previous studies along this line of research;
4) We show that PACO-based hybrids provide much better results than other hybrids and the pure LS approaches.

The remainder of this paper is organized as follows. In Section II, we introduce the investigated LK and MNS algorithms as well as their new hybrid versions. We then present our experimental study and discuss its results in Section III. Finally, the paper ends with conclusions and plans for future work in Section IV.

II. Investigated Algorithms

LS algorithms start at a random or heuristically-generated solution. They remember the best solution discovered so far and try to improve it. These improvements usually take place in the form of modifications to a tour in the TSP context. The most prominent examples of such modifications are m-opt moves [13], which include the exchange of two cities

Proc. 2015 IEEE 14th Int'l Conf. on Cognitive Informatics & Cognitive Computing (ICCI*CC'15)
N. Ge, J. Lu, Y. Wang, N. Howard, P. Chen, X. Tao, B. Zhang, & L.A. Zadeh (Eds.)
978-1-4673-7290-9/15/$31.00 ©2015 IEEE

corresponding to the deletion and addition of four edges (4-opt) [16, 17], the rotation of a sub-sequence of cities to the left or right resulting in the deletion and addition of three edges (3-opt) [16, 18], while the reversal of a sub-sequence of a tour requires deleting and adding two edges (2-opt) [16, 19]. If the LS concludes that it cannot further improve its best tour through the modifications it is able to conduct, it may apply a random modification to the tour in order to escape from the dead-end (while remembering the best overall solution in an additional variable). This process is repeated until a termination criterion is reached.

A. The Lin-Kernighan Algorithm

The LK algorithm for solving TSPs is an LS approach published by Lin and Kernighan [4] in 1973. Its derivatives dominate today's TSP research. Although the LK algorithm cannot provide general upper-bound guarantees for the solution quality, it achieves excellent results in practical scenarios. Many improvements have been proposed and many algorithms adopt the idea of LK. For example, when the Chained Lin-Kernighan (CLK) algorithm [20] arrives at a local optimum from which it cannot escape, it generates a new solution by a random 4-opt move instead of restarting at a random solution. CLK performs particularly well on TSPs with a large number *n* of cities. Meanwhile, the Lin-Kernighan-Helsgaun (LKH) algorithm [21, 22] may be the most efficient LK variant. LKH changes the step width of LK from 2 to 5.

A k-opt LS algorithm iteratively replaces k edges from the current best tour with another k edges to generate a new shorter tour. A tour can be considered as k-optimal when it is impossible to further improve the quality of the tour in this way. The larger the value of k, the more likely it is that the final tour is optimal. An *n*-optimal tour is therefore necessarily optimal. However, the runtime needed to test all k-exchanges increases rapidly as the number of *n* cities increases. As a result, 2-opt and 3-opt moves are most commonly used.

LK can be considered as a variable k-opt LS. At each step, the algorithm tests, for ascending values of k, whether replacing k edges may achieve a shorter tour. The algorithm works as follows. Let T be the current tour and N the node set containing all cities. The algorithm will then, in each iteration, construct the sets $X = \{X_1, \ldots, X_k\}$ of edges to be deleted from T and $Y = \{Y_1, \ldots, Y_k\}$ the set of edges to be added to T, such that the resulting tour would be valid and shorter. The interchange of these edges is then a k-opt move. In the beginning, X and Y are empty. Pairs of edges are added to X and Y such that the end node of the edge added to X is the starting node of the edge added to Y, whose end node will then become the starting node of the edge added to X in the next iteration, if any.

A necessary but not sufficient condition that the exchange of edges in X and Y results in a valid tour is that the chain is closed, i.e., that the end node of Y_k is the start node of X_1. The sets X and Y must further be disjoint, i.e., no added edge is deleted again and no deleted edge is added.

Algorithm 1 Lin-Kernighan Algorithm

Input: A random initial tour T_s
Output: The shortest tour T^* and its length L^*

1: Set a random initial tour T_s as the current tour T
2: Set $i = 1$. Choose a node as t_1. t_1 is the start node of the entire search procedure. If every node has already been tested as t_1, go to step 1.
3: Choose an edge $X_1 = (t_1, t_2)$ that belongs to T. If all edges have been tried as X_1, go to step 2.
4: Choose an edge $Y_1 = (t_2, t_3)$ that does not belong to T so that $G_1 = g_1 > 0$. If this is impossible and all choices for Y_1 have been tested, go to step 3.
5: Set $i = i + 1$;
6: Choose $X_i = (t_{2i-1}, t_{2i})$, such that:
a. $X_i \neq Y_p$, for all $p < i$, and
b. Add an edge between t_{2i} and t_1 so that $T \cup Y \setminus X$ can form a tour T'.
If T' is a shorter tour than T, set T = T' and $T^* = T'$, let L^* = the length of T^*, go to step 2.
If $i = 2$ and all choices for X_i have been tested, set $i = 1$, go to 4.
If $i > 2$ and all choices for X_i have been tested, set $i = 2$, go to step 7.
7: Choose $Y_i = (t_{2i}, t_{2i+1})$, such that:
a. $G_i = g_1 + \ldots + g_i > 0$,
b. $Y_i \neq X_p$ for all $p \leq i$, and
c. X_{i+1} exists.
If Y_i can be chosen, go to step 5.
If $i = 2$ and all choices for Y_i have been tested, go to step 6.
If $i > 2$ and all choices for Y_i have been tested, set $i = 2$ and go to step 7.
8: Stop (or start with a new initial solution)

Suppose $g_i = D_{start(X_i),end(X_i)} - D_{start(Y_i),end(Y_i)}$ is the gain of deleting edge X_i and adding Y_i. G_i be the sum $g_1 + \ldots + g_i$. In every step i, Y_i must be chosen so that G_i is positive. This criterion seemed restrictive, but it is actually not, because if the sum of a sequence of numbers is positive, there is a cyclic permutation of these numbers such that every partial sum is positive.

Finally, for any $i \geq 3$, the edges X_i are chosen such that a tour can be achieved by linking the end of X_i to the start of X_1. This reduces the runtime and simplifies the code.

Backtracking is allowed only for $i \in \{1, 2\}$: If no improvement has been found for $i > 2$, i is reset to 2 (and X and Y are reset accordingly). Then, the next candidate edge for Y_2 is tested. After each choice for Y_2 has been exhausted, the algorithm backtracks to X_2. When all choices for X_2 have been tested without success, it tracks back to Y_1 and finally to X_1. If even then no improvement can be discovered, the original LK restarts at a new random tour.

In our implementation of the algorithm, we replace the current tour with a new tour as soon as an improvement was

found. This is different from the original algorithm, which continues its steps by recording potential exchanges to find an even shorter tour until no more exchanges can be done, and then the current tour is replaced by the best tour discovered. This approach of the original algorithm complicates the code and can neither give a better solution quality nor reduce its runtime [21].

The LK algorithm is sometimes implemented by using candidate sets, i.e., by limiting the choices of neighbors in a tour for any given node. This restriction is omitted in our implementation. Finally, instead of restarting at an entirely random tour, our implementation uses the method described in [13], where a randomly chosen sub-sequence of the current tour is randomly shuffled.

B. Multi-Neighborhood Search

Another efficient LS method for the TSP is the MNS algorithm. In each iteration, MNS performs an $\mathcal{O}(n^2)$ scan that can investigate four neighborhoods (city swap, sub-sequence rotate left, sub-sequence rotate right, and reverse) of a tour at once. It tests all indexes i and j as potential indexes for cities to swap or start and end indexes of sub-sequence rotations and reversals. For each pair $\{i, j\}$, the gain is computed and all discovered improving moves enter a queue. The access to the distance matrix D is minimized by remembering (and updating) the lengths of all n edges in the current tour and avoiding the checking of redundant moves (swapping the cities at index i and $i + 1$ is equivalent to a reversal of the sub-sequence from i to $i + 1$, for instance). After the scan, the best discovered move is carried out. Doing this may invalidate some other moves in the queue, e.g., if a sub-sequence reversal that overlaps with a potential sub-sequence left rotation was performed. After pruning all invalidated moves from the queue, the remaining best move is carried out, if any. If the queue becomes empty, another scan of the current solution is performed, as new moves may have become possible. During this scan, only moves that at least intersect with the previously modified sub-sequence(s) of the current best solution need to be considered (to speed up the search). If no improving moves can be found anymore, a random sub-sequence of the current tour is randomly shuffled.

This algorithm has performed the best among all of the algorithms tested in [13]. While LK examines a sub-set of possible k-opt moves, MNS scans for all 2-opt and some 3- and 4-opt moves, i.e., is more limited in what it can do. Due to its structure, however, the search for moves can be done very quickly and multiple moves may be discovered at once. In this work, we will analyze which algorithm concept leads to better performance and under which circumstances.

C. Evolutionary Algorithms

EAs are the most well-known Evolutionary Computation (EC) methods [23, 24]. EAs first generate a set of λ random solutions. Out of these, the best $\mu \leq \lambda$ solutions will be selected as "parents" of the second generation: λ offspring solutions are created by applying either a unary (mutation) or binary (crossover) operator to the parents. From then on, the μ best individuals are selected from the λ offspring solutions and their μ parents in each generation in the case of a $(\mu+\lambda)$-EA. A (μ, λ)-EA selects only from the λ offspring. In this paper, we investigate such EAs using the four neighborhoods also used by MNS as mutation operators. Edge Crossover [25], which generates a new solution by picking edges belonging to either of its two parents, is applied as a recombination operator at a crossover rate of $1/3$. It is considered to be one of the best crossover operators for the TSP [13].

Hybridization of EAs with LS has a long tradition [26]. Such hybrid approaches, where the LS algorithm either takes the place of the mutation operator or is applied to each new solution (stemming from either mutation or crossover), are called Memetic Algorithms (MAs). Especially in the TSP domain, MAs have been performing well in general. We therefore propose two MAs: hMA($\mu \dagger \lambda$)-LK and hMA($\mu \dagger \lambda$)-MNS. The 'h' in the name indicates that the first population of the MAs is not generated randomly, but instead stems from the Edge-Greedy, Double Minimum Spanning Tree, Savings, Double-Ended Nearest Neighbor, and Nearest Neighbor Heuristic, as in [13].

D. Ant Colony Optimization

The ACO algorithm is another EC approach and was first introduced by Dorigo in 1992 [8]. It took inspiration from the way ants find and reinforce short paths during foraging using pheromones for communication. Although such algorithms would typically perform well in many small-scale combinatorial problems, they suffer from quadratic memory requirements as well as quadratic complexity in the process of creating solutions.

In this paper, we consider a state-of-the-art ACO variant, PACO [15], which has linear memory requirements. The PACO algorithm maintains a population of n solutions. Pheromones are defined by the edges occurring in these solutions. In each algorithm iteration, m solutions are created like the standard ACO and the "oldest" solution in the population is replaced by the best of the newly generated solutions. Limited hybrid ACO approaches have been applied to the TSP, although it was shown in [13] that they perform particularly well. We therefore propose hybrid hPACO(m, n)-MNS and hPACO(m, n)-LK algorithms, which are heuristically initialized in the same way as the hMAs.

III. Experiments and Results

A. Experimentation with Anytime Algorithms

Most metaheuristics, including EAs, MAs, ACO, as well as all LS methods, are anytime algorithms [27]. Even several exact methods, such as BB algorithms [28], fall into this category. Anytime algorithms can provide a best guess of what the optimal solution of a problem could be at *any time* during their run. LK and MNS begin with a random solution and iteratively refine it. In a TSP, BB algorithms maintain the best solution discovered so far and investigate a set of other

solutions only if the lower bound for their tour length is better than the actual tour length of that best-so-far solution.

If an algorithm $\mathcal{A}$ provides a better final solution than another algorithm $\mathcal{B}$, does this make algorithm $\mathcal{A}$ better? The traditional answer would be *yes*, but what if the best guess of $\mathcal{A}$ for the solution is much worse than $\mathcal{B}$'s, except after a very long runtime? Thus, due to their nature, anytime algorithms cannot just be characterized by a final solution and runtime requirement, but only by their whole runtime behavior [13].

Experiments for analyzing the behavior of an algorithm over runtime are important but also cumbersome. They generate much data and manual evaluation can take more time than implementing the algorithm itself.

One of the first frameworks aiming to reduce the workload of an experimenter is the COmparing Continuous Optimizers (COCO) [29] system for numeric optimization. It helps to automatize most of the steps involved. Necessary data is automatically collected from automatically executed experiments. A statically structured paper that contains diagrams with runtime behavior information is generated after an automated evaluation procedure.

UBCSAT [30], an experimental framework for satisfiability (SAT) problems, focuses on a specific algorithm family, the stochastic LS. UBCSAT gathers information by a trigger architecture, which is able to compute complex statistics and provide them to the running algorithm. COCO and UBCSAT can both measure algorithm performance over runtime instead of just focusing on the end results.

Our recently developed *TSP Suite* [13] focuses on investigation of TSP solvers. Like COCO, data is collected during the evaluation of candidate solutions. However, the reports are not statically structured papers. Instead, they are more like small theses and can be freely configured. They contain in-depth descriptions of the experimental procedure and give several different statistical analyses such as statistical tests comparing measured runtimes and end results, automated comparisons of estimated running time (ERT) [29] curves over goal objective values or problem scales and automated comparisons of empirical cumulative distribution functions (ECDFs) [29–31]. All of the information is aggregated into human-readable conclusions about the algorithm performance in the form of global rankings. The *TSP Suite* further provides JUnit tests to validate implemented algorithms. Algorithms can be implemented for normal, sequential execution, but the framework is still able to parallelize and distribute the workload on a cluster (without requiring any additional software).

The *TSP Suite* is the first framework addressing the issue of measuring runtime. Measuring runtime in CPU seconds, for instance, produces machine-dependent results. Even if normalized runtimes (*NT*) are calculated based on machine performance factors, they remain problem specific and may not represent the utility of black-box metaheuristics in general. Counting the number of generated solutions (i.e., objective function evaluations, or *FE*s in short) is the most-often used alternative in benchmarking. However, it neglects the fact that one *FE* in ACO, for instance, has quadratic complexity whereas in a LS algorithm, it may be in $\mathcal{O}(1)$ on the TSP. The *TSP Suite* thus measures runtime in four different measures, CPU time, normalized CPU time, FEs, and the number *DE* of accesses to the distance matrix D of a TSP, in order to provide a balanced overview on algorithm performance.

B. Experimental Setup

Our experiments were conducted using the symmetric *TSPLib* [32] benchmark cases, for which all optima are known. We thus can measure the quality of a solution as relative error f, i.e., the factor by which a solution is "longer" than the optimum. $f = 0$ stands for the optimal solution, $f = 1$ indicates that it is twice as long. We obtained and evaluated results for all 110 instances in the *TSPLib*. The scale *n* of the largest instance in our study is 85,900, which is much more than the largest scale of the original LK experiments in [4], where a 318-city problem is the largest one.

For the algorithms, 15 LK setups were built: The pure LK, six hybrids with PACO, and eight hybrid MAs. The same numbers of setups and configurations of MNS were tested. Additionally, we investigated the pure EA(128+256), the pure PACO(3,25), as well as the heuristically initialized hEA(128+256) and hPACO(3,25).

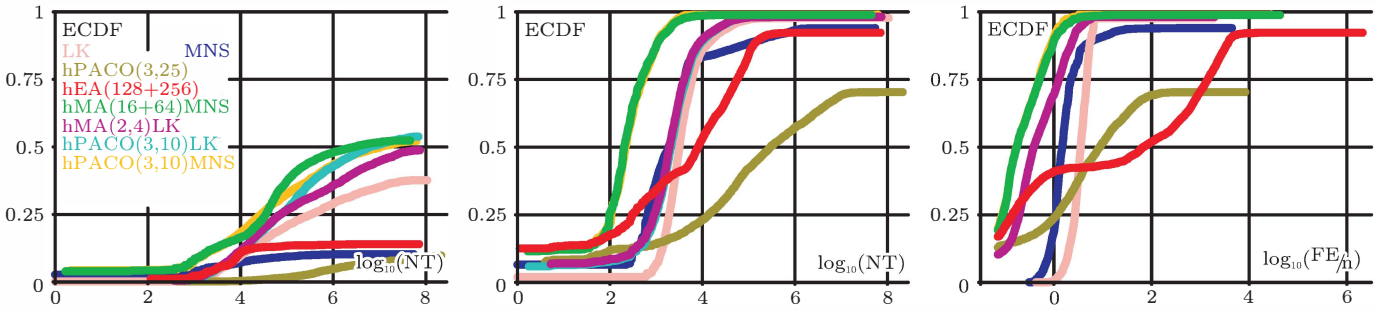
C. Pure Algorithm Performance

Let us first explore the performance of the pure LK algorithm and compare it with MNS and the pure EC methods. According to the automated ranking obtained from the *TSP Suite*, LK has the best performance amongst these algorithms.

In Figure 1, we plot the ECDF for different goal errors F_t and runtime measures. The ECDF illustrates the fraction of runs that have discovered a (best) solution with $F_b \leq F_t$. Hence, an algorithm is good if its ECDF comes as close to 1 and as soon as possible. The pure EA and pure PACO always have the worst performance among the algorithms considered. We therefore do not include them in our figures for the sake of readability.

The ECDF of LK in Figure 1a, based on the normalized CPU runtime measure *NT*, increases smoothly and slowly and approaches 0.4 but never reaches it for $F_t = 0$. In other words, a global optimum can be reached in about 40% (of the runs) of all benchmark cases under the given computational budget. Although this does not seem to be very good, LK still has the best performance among all pure algorithm settings. The ECDF of MNS increases slightly faster than LK's in the beginning, but later slows down and finally reaches a little more than 0.1. Only for small time budgets, MNS outperforms LK, which can solve four times as many problems if given enough time.

In the other two sub-figures of Figure 1, we increase the goal error F_t to 0.1, i.e., to investigate the fraction of runs for finding a solution that is up to 10% longer than the optimum. Again, the ECDF curves intersect. LK can reach the goal in almost all of the problems while MNS can solve about 95% of them. With the increase of F_t, the gap between the two algorithms becomes smaller.



(a) ECDF for *NT* and $F_t = 0.0$. (b) ECDF for *NT* and $F_t = 0.1$. (c) ECDF for *FE*s and $F_t = 0.1$.

Fig. 1: ECDF diagrams for different (log-scaled) runtime measures and goal errors.

If we measure time in terms of *FE*s (Figure 1c) instead of CPU seconds, MNS performs better within a relative small time budget and rises earlier. After the intersection, the ECDF of LK increases faster than the one of MNS and reaches a higher level in the end.

We can also compare the two pure LS methods with two of the best setups of pure EC methods and their variants using initialization heuristics. The EA(128+256) always performs worse than LK. Using initialization heuristics provides an initial advantage for the hEA(128+256), but it is still overtaken by LK in the end. The setups with PACO exhibit similar behaviors.

From the results, we can conclude that: *1)* There is a threshold of computational budget before which MNS has a higher chance to reach the global optimum and above which LK has a higher chance to reach the global optimum; *2)* The ECDF of LK always has a higher end point than the one of MNS.

We now analyze the relationship between the algorithm performance and problem scale n. To do so, we grouped the problems by their scale according to the different powers of two.

From Figure 2, which illustrates the best objective value F_b discovered by an algorithm over runtime, we see that LK can find good solutions for instances with n up to 511 quite rapidly. With the increasing of instance scale, the performance of LK decreases, but it can still find approximate solutions with $F_t \leq 0.05$ for $512 \leq n \leq 32767$. For example, on an instance with 18512 cities, a solution with f of around 0.03 is discovered.

Compared to LK, MNS can quickly find optimal solutions with n between 14 and 63. MNS performs better in smaller instances with n between 8 and 31. When n increases, MNS performs relatively worse compared to LK. For n between 32 and 1023, after a rapid improvement of the tour length in the beginning, the progress of MNS becomes much slower and it is outperformed by LK. For n between 1023 and 65535, LK always outperforms MNS.

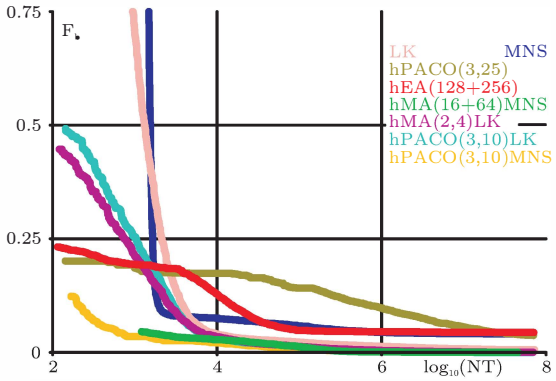
The pure EC methods all create (or receive from the heuristics) better solutions early on, but their progress is much slower than the LS algorithms. From Figure 2, we can see that LK still has the best end results.

We conclude that on small-scale problems, MNS is better. For moderate-size problems, MNS performs better initially but is later outperformed by LK, while for large-scale problems LK always performs better. Such findings, again, can only be made by using statistics over runtime instead of just comparing final results, as it is (unfortunately) commonly done in the literature.

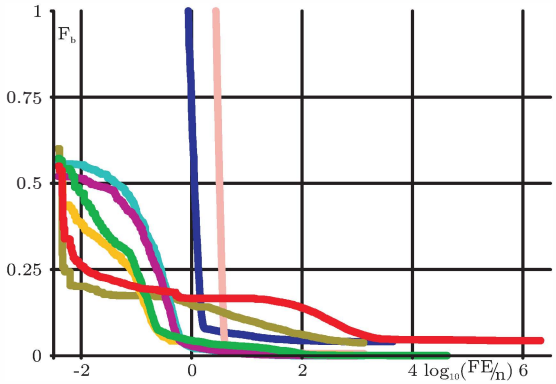
While the *NT* is related to the actual consumed runtime, we can also measure the progress in terms of *FE*s, which is related to the number of candidate solutions constructed. We therefore include two diagrams based on *FE*s in Figure 2 as well. Although LK still has better end results, MNS is now in the lead for a much longer time (i.e., if time is measured in *FE*s). This may be due to the fact that MNS always executes the best improvement move found in a whole scan of the solution while LK takes the first improvement move that it finds and, thus, is more likely to have many smaller improvements (each needing 1 *FE*).

In Figure 3, we plot the ERT in terms of *NT* (y-axis) for a given solution quality threshold F_t (x-axis), i.e., the estimated normalized runtime it will take for an algorithm to reach F_t. For smaller F_t, the time increases. In terms of the ERT measured by counting accesses to the distance matrix as a time unit (*DE*s, not illustrated), LK is always better than MNS. In terms of *FE*s, MNS seems to perform better with $F_t > 0.14$ and is worse otherwise. In terms of *NT*, the situation changes again and there are two intersections of the ERT curves: MNS performs better for $0.1 \leq F_t \leq 0.5$ and LK is better otherwise. Thus, if different time measures were used, different observations could be found.

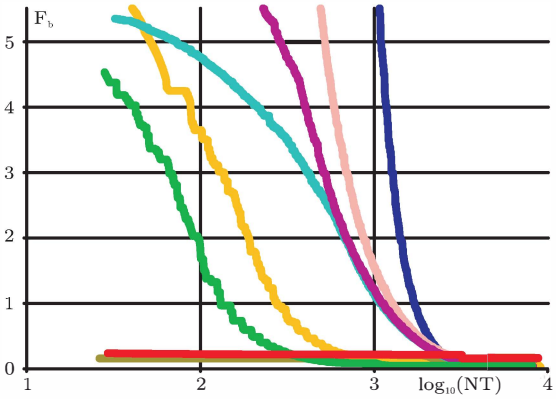
From all of the above, we conclude that the tested LK algorithm is better than MNS in most situations and both the LS algorithms are much better than the pure EC methods. LK can also outperform the EC methods that use heuristics for initialization, while pure, uninitialized MNS cannot. Consequently, the *TSP Suite* ranks the algorithms as follows: LK, hEA(128+256), hPACO(3,25), MNS, PACO(3,25), and EA(128+256).



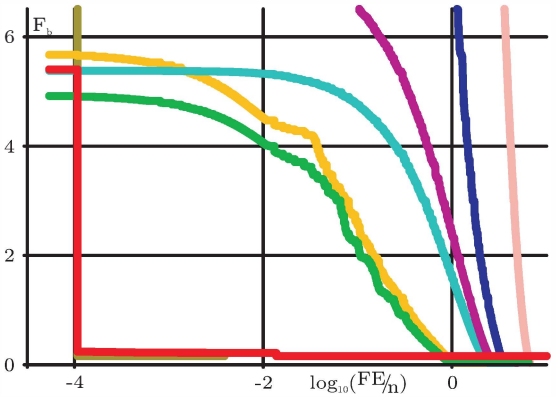
(a) Progress in F_b over NT for $256 \leq n < 512$.



(b) Progress in F_b over FEs for $256 \leq n < 512$.



(c) Progress in F_b over NT for $16384 \leq n < 32768$.



(d) Progress in F_b over FEs for $16384 \leq n < 32768$.

Fig. 2: Progress diagrams for different (log-scaled) time measures and problem scales.

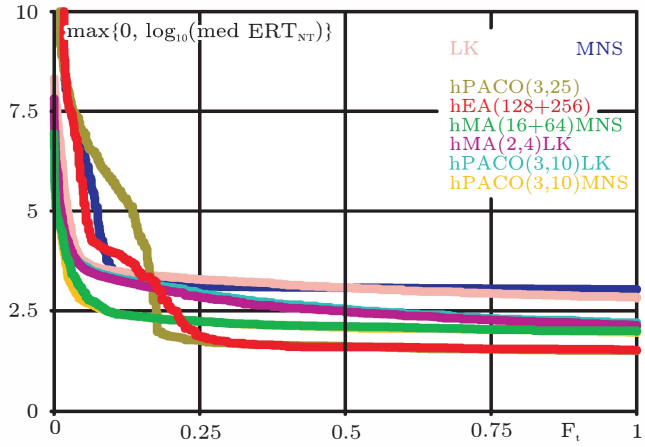


Fig. 3: (Log-scaled) ERT in terms of NT.

D. Hybrid Algorithm Performance

We also investigated our newly proposed hybridized versions of LK and MNS with EAs, i.e., MAs, and PACO. Each of these four hybrids was tested using six different settings for PACO and eight for the MAs.

From the complete ranking generated by the *TSP Suite* based on all setups, it can be clearly seen that different setups of the same corresponding algorithms have relatively similar behavior, no matter what kind of measure is considered (e.g., hPACO(3,10)-MNS and hPACO(5,10)-MNS have similar behavior in terms of ECDF, ERT and progress). Four of the best setups from different categories, i.e., hPACO(3,10)-LK, hMA(2,4)-LK, hPACO(3,10)-MNS and hMA(16+64)-MNS, have been chosen for illustrating the figures.

From Figure 1, we can see a significant improvement of LK after being hybridized with EAs and PACO. The hPACO(3,10)-LK and hMA(2,4)-LK outperform pure LK on both the speed and end results. We also see that the ECDF curves of hPACO(3,10)-LK and hMA(2,4)-LK start slightly earlier than those of pure LK, increase more rapidly, and finally reach higher (better) end points. Pure LK is able to solve about 40% of the benchmark instances, while hPACO(3,10)-LK can almost solve 55% and hMA(2,4)-LK 48%, making PACO the better global optimization method to hybridize LK with than EAs.

The same observations can be made with the MNS hybrids: The hPACO(3,10)-MNS and hMA(16+64)-MNS largely outperform pure MNS. Their ECDF curves increase rapidly and reach at almost 52%, which is five times as much as what pure MNS can achieve. There is always a significant ECDF gap between the hybrid and pure algorithms.

The hybrids based on MNS are better in the beginning, while hPACO(3,10)-LK can reach a higher end point. hMA(2,4)-LK is always worse than the hybrids of MNS. When we measure runtime in FEs, similar behavior can be observed.

When we set the goal error as $F_t = 0.1$, the hybrid algorithms again outperform the pure ones. The PACO hybrids are slightly better than the MAs. However, now the MNS

hybrids are always better than those of LK. Their ECDF curves (for $F_t = 0.1$) increase earlier, more rapidly, and finally reach higher end points.

The ERT diagrams in terms of *DE*s and *NT* for a given F_t share similar shapes with those in Figure 3. In all of them, hybrid MNS variants can find good solutions the fastest, followed by the hybrid LK algorithms. Pure LS and EC methods are much slower. From Figure 2, we again confirm that the MNS hybrids are initially faster than the hybrid LK algorithms, although LK hybrids later on find better-quality solutions.

For large-scale problems, the heuristically initialized EC methods start with much better solutions, even better than the hybrid algorithms. This, at first glance, may be confusing, since our hybrid algorithms were also heuristically initialized, with the same heuristics. However, for each generated solution, the hybrid algorithms will apply their respective LS procedure first before entering the solution into the population and creating the next initial solution. This means that the hEA and hPACO have more heuristic results earlier, while the hMA and hybrid hPACO variants with LS first spend more time on searching locally.

The most interesting finding, however, is that hybrid MNS methods tend to outperform hybrid LK approaches, although pure MNS is clearly worse than pure LK. While an almost "additive" effect of hybridization was observed in [13], i.e., a better LS algorithm hybridized with a better global search method leads to a better hybrid approach, here we have a contrasting observation. Although hybrid LK can find better solutions on the long run, the efficient main loop of MNS, which can be implemented in a compact way, makes use of a (linear-sized) cache, and can discover several improvements at once, thus provides faster convergence. Since the maximum runtime granted to a run executed with the *TSP Suite* is one hour and in practical scenarios much less time is usually available, MNS appears to be the better choice for hybridization. On large-scale instances, for a fraction of the runtime, it may produce solution approximations that are shorter by two times the optimal length than those discovered by hybrid LK. Finding such as this would not be possible by just collecting end-of-run results, as by these hybrid LK would look better.

Regardless of what runtime is available, we confirm that hybrid algorithms are better than both pure LS and pure EC methods, heuristically initialized or not. The aggregated algorithm ranking provided by the *TSP Suite* when comparing all setups in regard to ECDF, ERT, final results, expected runtime to the optimum, and progress according to different runtime measures is

hPACO(3,10)-MNS, hPACO(5,10)-MNS,
hPACO(3,25)-MNS, hPACO(5,25)-MNS,
hPACO(10,10)-MNS, hPACO(10,25)-MNS,
hMA(16+64)-MNS, hPACO(3,10)-LK, hPACO(3,25)-LK,
hMA(16,64)-MNS, hPACO(5,10)-LK, hMA(128+256)-MNS,
hMA(2+4)-MNS, hPACO(5,25)-LK, hMA(2+8)-MNS,
hMA(2,4)-LK, hPACO(10,10)-LK, hMA(2,8)-LK,
hMA(2,8)-MNS, hPACO(10,25)-LK, hMA(16,64)-LK,
hMA(16+64)-LK, hMA(2+8)-LK, hMA(128,256)-MNS,
hMA(128+256)-LK, hMA(2,4)-MNS, hMA(2+4)-LK,
hMA(128,256)-LK, LK, hEA(128+256), MNS,
hPACO(3,25), PACO(3,25), and EA(128+256).

This ranking also reveals that PACO seems to be a better EC method to hybridize with than an EA (a hybrid EA is an MA). This is exactly the same observation previously seen in [13].

IV. Conclusions and Future Work

In this work, we have presented a large-scale experimental study in which several variants of LS and EC algorithms as well as their newly proposed hybrids were applied to all 110 symmetric instances of the *TSPLib*, 30 runs per setting, each limited to one hour of runtime. Our experiments have led us to four major conclusions:

1) The pure LK algorithm works well on small- and medium-scale instances. It can also find approximate solutions in good qualities for large-scale instances.
2) The pure LK algorithm is better than MNS and EC methods. Sometimes, MNS and some of the EC methods can provide better solutions at the beginning, but are eventually overtaken by LK.
3) Our proposed hybrid algorithms have much better performance than the pure LS and EC approaches.
4) Based on the same LS algorithm, the best setup of hybrid PACO is always better than any setup of hybrid EAs. The hybrids of LK with PACO are the best variants among other LK-based algorithms.
5) Although pure LK is better than MNS, hybrid MNS outperforms hybrid LK.

The last point is particularly interesting and deserves further exploration. In our future work, we will investigate hybrid EC-LK/MNS methods that can use both LK and MNS as their LS. That is, before refining a solution, we could randomly select which LS algorithm to apply. The random distribution could change over time, starting mainly with MNS and later switching more regularly to LK, in order to fully utilize the initial high speed of MNS hybrids while also leverage the better end-result quality provided by LK hybrids. We are also interested to investigate Tabu Search variants and compare them with the algorithms presented here.

Acknowledgments: We acknowledge support from the Fundamental Research Funds for the Central Universities, the National Natural Science Foundation of China under Grant 61150110488, Special Financial Grant 201104329 from the China Postdoctoral Science Foundation, the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01, the European Union 7th Framework Program under Grant 247619, and the University of Newcastle's Faculty of Science and Information Technology Strategic Initiatives Research Fund under Grant 1031415. The experiments reported in this paper were executed on the supercomputing system in the Supercomputing Center of University of Science and Technology of China.

References

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, ser.

Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, February 2007. [Online]. Available: http://books.google.de/books?id=nmF4rVNJMVsC

[2] E. L. G. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, ser. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, September 1985. [Online]. Available: http://books.google.de/books?id=BXBGAAAAYAAJ

[3] G. Z. Gutin and A. P. Punnen, Eds., *The Traveling Salesman Problem and its Variations*, ser. Combinatorial Optimization. Norwell, MA, USA: Kluwer Academic Publishers, 2002, vol. 12. [Online]. Available: http://books.google.de/books?id=TRYkPg_Xf20C

[4] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research (Oper. Res.)*, vol. 21, no. 2, pp. 498–516, March–April 1973. [Online]. Available: https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic

[5] T. Weise, *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published), 2009. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[6] T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation*, ser. Computational Intelligence Library. New York, NY, USA: Oxford University Press, Inc., Dirac House, Temple Back, Bristol, UK: Institute of Physics Publishing Ltd. (IOP), and Boca Raton, FL, USA: CRC Press, Inc., January 1, 1997. [Online]. Available: http://books.google.de/books?id=n5nuiIZvmpAC

[7] K. A. De Jong, *Evolutionary Computation: A Unified Approach*, ser. Bradford Books. Cambridge, MA, USA: MIT Press, February 2006, vol. 4.

[8] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, Milano, Italy: Dipartimento di Elettronica, Politecnico di Milano, January 1992, in Italian.

[9] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization – artificial ants as a computational intelligence technique," *IEEE Computational Intelligence Magazine (CIM)*, vol. 1, no. 4, pp. 28–39, November 2006. [Online]. Available: http://iridia.ulb.ac.be/~mbiro/paperi/IridiaTr2006-023r001.pdf

[10] L. M. Gambardella and M. Dorigo, "Solving symmetric and asymmetric tsps by ant colonies," in *Proceedings of IEEE International Conference on Evolutionary Computation (CEC'96)*, K. Jidō and S. Gakkai, Eds. Nagoya, Aichi, Japan: Nagoya University, Symposium & Toyoda Auditorium: Los Alamitos, CA, USA: IEEE Computer Society Press, May 20–22, 1996, pp. 622–627.

[11] J. D. C. Little, K. G. Murty, D. W. Sweeny, and C. Karel, "An algorithm for the traveling salesman problem," Cambridge, MA, USA: Massachusetts Institute of Technology (MIT), Sloan School of Management, Sloan Working Papers 07-63, March 1, 1963.

[12] S. Dhakal and R. Chiong, "A hybrid nearest neighbour and progressive improvement approach for travelling salesman problem," in *Proceedings of International Symposium on Information Technology (ITSim)*, vol. 1, August 2008, pp. 140–143.

[13] T. Weise, R. Chiong, K. Tang, J. Lässig, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Computational Intelligence Magazine (CIM)*, vol. 9, no. 3, pp. 40–52, August 2014. [Online]. Available: http://www.it-weise.de/documents/files/WCTLTCMY2014BOAAOSFFTTSP.pdf

[14] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, "Variable neighbourhood search: Methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, March 1, 2010.

[15] M. Guntsch and M. Middendorf, "Applying population based aco to dynamic optimization problems," in *From Ant Colonies to Artificial Ants – Proceedings of the Third International Workshop on Ant Colony Optimization (ANTS'02)*, ser. Lecture Notes in Computer Science (LNCS), M. Dorigo, G. A. Di Caro, and M. Samples, Eds., vol. 2463/2002. Brussels, Belgium: Berlin, Germany: Springer-Verlag GmbH, September 12–14, 2002, pp. 111–122.

[16] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Journal of Artificial Intelligence Research (JAIR)*, vol. 13, no. 2, pp. 129–170, April 1999.

[17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Germany: Springer-Verlag GmbH, 1996. [Online]. Available: http://books.google.de/books?id=vlhLAobsK68C

[18] D. B. Fogel, "An evolutionary approach to the traveling salesman problem," *Biological Cybernetics*, vol. 60, no. 2, pp. 139–144, December 1988.

[19] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[20] D. L. Applegate, W. J. Cook, and A. Rohe, "Chained lin-kernighan for large traveling salesman problems," *INFORMS Journal on Computing (JOC)*, vol. 15, no. 1, pp. 82–92, Winter 2003.

[21] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," Roskilde, Denmark: Roskilde University, Department of Computer Science, Datalogiske Skrifter (Writings on Computer Science) 81, 1998.

[22] K. Helsgaun, "General k-opt submoves for the lin–kernighan tsp heuristic," *Mathematical Programming Computation*, vol. 1, no. 2-3, pp. 119–163, October 2009.

[23] C. Blum, R. Chiong, M. Clerc, K. A. De Jong, Z. Michalewicz, F. Neri, and T. Weise, "Evolutionary optimization," in *Variants of Evolutionary Algorithms for Real-World Applications*, R. Chiong, T. Weise, and Z. Michalewicz, Eds. Berlin/Heidelberg: Springer-Verlag, 2011, ch. 1, pp. 1–29. [Online]. Available: http://www.it-weise.de/documents/files/BCCDJMNW2011EO.pdf

[24] R. Chiong, F. Neri, and R. I. McKay, "Nature that breeds solutions," in *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering*, R. Chiong, Ed. Hershey, PA, USA: Information Science Reference, 2009, ch. 1, pp. 1–24.

[25] L. D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, J. D. Schaffer, Ed. Fairfax, VA, USA: George Mason University (GMU): San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., June 4–7, 1989, pp. 133–140.

[26] R. Chiong and P. Siarry, "Local search for real-world scheduling and planning," *Eng. Appl. of AI*, vol. 25, no. 2, pp. 207–208, 2012.

[27] M. S. Boddy and T. L. Dean, "Solving time-dependent planning problems," Providence, RI, USA: Brown University, Department of Computer Science, Tech. Rep. CS-89-03, February 1989.

[28] Y. Jiang, T. Weise, J. Lässig, R. Chiong, and R. Athauda, "Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the tsp," in *Proceedings of the IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS'14), Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI'14)*. Orlando, FL, USA: Caribe Royale All-Suite Hotel and Convention Center: Los Alamitos, CA, USA: IEEE Computer Society Press, December 9–12, 2014. [Online]. Available: http://www.it-weise.de/documents/files/JWLCA2014CAHBABAWECMLSATHOTT.pdf

[29] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking: Experimental setup," Orsay, France: Université Paris Sud, Institut National de Recherche en Informatique et en Automatique (INRIA) Futurs, Équipe TAO, Tech. Rep., March 24, 2012.

[30] D. A. D. Tompkins and H. H. Hoos, "Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat," in *Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, ser. Lecture Notes in Computer Science (LNCS), H. H. Hoos and D. G. Mitchell, Eds., vol. 3542. Vancouver, BC, Canada: Berlin, Germany: Springer-Verlag GmbH, May 10–13, 2004, pp. 306–320.

[31] H. H. Hoos and T. Stützle, "Evaluating las vegas algorithms – pitfalls and remedies," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, G. F. Cooper and S. Moral, Eds. Madison, WI, USA: San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., July 24–26, 1998, pp. 238–245, also published as Technical Report "Forschungsbericht AIDA-98-02" of the Fachgebiet Intellektik, Fachbereich Informatik, Technische Hochschule Darmstadt, Germany.

[32] G. Reinelt, "Tsplib – a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, Fall 1991.