Hybrid Ejection Chain Methods for the Traveling Salesman Problem

Weichen Liu[1], Thomas Weise[1,2], Yuezhong Wu[1], and Raymond Chiong[3]

[1] 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: lwc@mail.ustc.edu.cn, tweise@ustc.edu.cn, yuezhong@mail.ustc.edu.cn
[2] corresponding author
[3] School of Design, Communication and IT, Faculty of Science and IT, The University of Newcastle, Callaghan, NSW 2308, Australia. Email: Raymond.Chiong@newcastle.edu.au

Abstract. Local search such as Ejection Chain Methods (ECMs) based on the stem-and-cycle (S&C) reference structure, Lin-Kernighan (LK) heuristics, as well as the recently proposed Multi-Neighborhood Search (MNS), are among the most competitive algorithms for the Traveling Salesman Problem (TSP). In this paper, we carry out a large-scale experiment with all 110 symmetric instances from the *TSPLib* to investigate the performances of these algorithms. Our study is different from previous work along this line of research in that we consider the entire runtime behavior of the algorithms, not just their end results. This leads to one of the most comprehensive comparisons of these algorithms to date. We introduce a new, improved S&C-ECM that can outperform LK and MNS. We then develop new hybrid versions of our ECM implementations by combining them with Evolutionary Algorithms and Population-based Ant Colony Optimization (PACO). We compare them to similar hybrids of LK and MNS. Our results show that hybrid PACO-S&C, PACO-LK and PACO-MNS are all very efficient. We also find that the full runtime behavior comparison provides deeper and clearer insights, while focusing on end results only would have led to a misleading conclusion.

Keywords: Traveling Salesman Problem, Ejection Chain Methods, Lin-Kernighan Heuristic, Multi-Neighborhood Search, Hybrid Algorithms

1 Introduction

The Traveling Salesman Problem (TSP) [1, 14, 22] is a well-known $\mathcal{NP}$-hard problem in the field of combinatorial optimization. The problem can be stated

as follows: Given n cities (named from 1 to n) and the distances $D_{i,j}$ (with $i, j \in 1, 2, \ldots n$) between them, a salesman starts from one city, visits each of the cities once, and then returns to the original city. The assignment is to find the order in which the salesman should visit the cities with the shortest overall travel distance. A tour, i.e., a candidate solution to the TSP, can be defined as permutation $t = (t_1, t_2, \ldots, t_n)$ of the cities to visit. The task is to find a t that minimizes the sum $D_{t[1],t[2]} + D_{t[2],t[3]} + \ldots + D_{t[n],t[1]}$. We focus on symmetric TSP instances in which $D_{i,j} = D_{j,i}$ holds.

The TSP is $\mathcal{NP}$-hard, therefore any existing exact TSP algorithms have exponential worst-case runtime complexity. To get good approximate solutions within acceptable time, many approaches have been introduced, including Local Search (LS) algorithms, Evolutionary Algorithms (EAs) [3, 5, 30], and Ant Colony Optimization (ACO) [6, 7, 9]. The state-of-the-art algorithms are two LS families: The more well-known is the Lin-Kernighan (LK) heuristic [23], while the other is the Ejection Chain Method (ECM) based on a stem-and-cycle (S&C) structure. ECMs are reported to provide better results than (pure) LK in several studies [26–28], but at the cost of more runtime. Recently, another competitive LS algorithm, the Multi-Neighborhood Search (MNS) [31], was introduced and found to be particularly suitable for hybridization [33].

In this paper, we introduce a new, improved ECM working on the S&C structure. We compare it with two existing S&C-ECMs and show that it significantly outperforms them. An in-depth and statistically sound comparison of all three S&C-ECMs with LK and MNS is then carried out, and the results show that our improved ECM outperforms both LK and MNS in terms of results and speed. We also hybridize S&C-ECMs with EAs and PACO, and compare them to similar hybrids based on LK and MNS. We conduct a large-scale experimental study and apply advanced, runtime-behavior based statistics that provide much more performance information of these algorithms than simple end-result comparisons. We confirm that PACO-based hybrids are much better than EA-based ones (i.e., Memetic Algorithms) and pure LS approaches. Interestingly, hybrid MNS is shown to be the most suitable LS for hybridization in our experiments, although it is outperformed by both LK and ECMs in its pure form. This study provides more precise, comprehensive, and statistically sound evaluations than any other previous work on this topic.

The remainder of this paper is organized as follows. First, we present the investigated S&C-ECMs, LK heuristic and MNS, as well as their new hybrid versions in Section 2. Since LK, ECMs based on the S&C structure and MNS mark the state-of-the-art in terms of LS for the TSP, this section indirectly also describes the related work. In Section 3, we discuss our experimental study and analyze its results. The paper ends with conclusions and plans for future work in Section 4.

2 Investigated Algorithms

Today, the best known algorithms for the TSP are LS methods and in this paper, we consider three of them. An LS algorithm maintains and iteratively tries to improve a single candidate solution. The initial solution is often randomly generated or stemmed from a simple heuristic. In each step, the LS explores the neighborhood of the solution, which is spanned by possible applications of a search operator. If it contains a better solution, then this solution is accepted as the basis for the next iteration. If no better solution is found, the LS either terminates or restarts. In the latter case, either a new, random solution is used or the current solution is randomly modified in a way that is beyond what the search operation could achieve. Restarts are necessary, because the neighborhoods spanned by search operators are much smaller than the search space. As a result, local optima, i.e., solutions that are not optimal but whose neighborhoods only contain inferior solutions, exist.

In the TSP, the most common search operations are k-opt moves, which delete k edges in a tour and replace them with k other edges [31]. A 2-opt move, replacing two edges, corresponds to the reversal of a part of the tour [18, 21]. Rotating a part of the tour one step to the left or right corresponds to a 3-opt move [8, 21]. Swapping two cities is a 4-opt move [21, 25].

2.1 ECMs

The ECM was introduced by Glover in 1992 [11]. ECMs provide k-opt moves for discrete optimization problems. The basic component of an ECM is the data structure it processes with its search operations. This structure can be different from simple path or adjacency list representations.

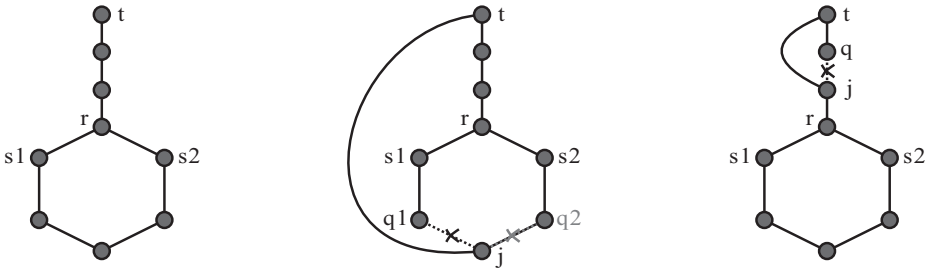
We investigate the S&C reference structure, which consists of a path (called stem) attached to a cycle of nodes, as illustrated in Figure 1a. The common node of S&C is called root r and its two adjacent nodes on the cycle are called sub-roots (s_1 and s_2). The root r marks one end of the stem. The other end is the tip t.

Only if the stem is degenerated to become a single node (i.e., $r = t$), the S&C structure is a tour. Otherwise, the S&C structure can be transformed to two candidate tours by removing the edge between one of the sub-roots si and the root r, and then re-connecting si to the tip t. The better one of these two trial solutions can be chosen.

The search does not take place in the space of possible tours, but directly on the S&C structures, which are iteratively refined according to a set of rules.

2.2 Fundamental S&C Approaches

The Fundamental S&C (`FSC`) algorithm proposed by Rego [27] defines a complete LS procedure including tabu criteria and concepts of limiting the search depth to improve performance. The core of `FSC` is Glover's two rules [12] for updating the S&C, i.e., the search operations of `FSC`:



(a) An example of a S&C reference structure.

(b) Rule 1 of the Fundamental S&C ECM.

(c) Rule 2 of the Fundamental S&C ECM.

Fig. 1: Examples of a S&C reference structure and the two rules of the Fundamental S&C ECM.

1. Choose a node j on the cycle. Let the two nodes adjacent to it be q_1 and q_2. Select one of them and refer to it as q. Delete any edge adjacent to j and then add edge (t, j).
2. Choose a node j on the stem. Let the node adjacent to j and farther away from r than the other adjacent node be called q. Connect t to j and delete edge (j, q).

In both cases, q becomes the new tip t (see Figures 1b and 1c). Each of their applications *ejects* a sequence of edges from a tour and replaces them. Based on these two rules, a S&C structure can be transformed to another one.

What Glover [10] did not specify, however, is how to choose j and how to iterate the rule application. Rego [27] therefore defined the equation $e = D_{add} - D_{delete}$, where D_{add} is the length of the edges to be added and D_{delete} the length of the deleted ones. `FSC` iterates over all the available j nodes in the S&C and applies both rules 1 and 2. From these choices, it identifies the move with the smallest corresponding value of e. By applying this move, a solution is changed to another one.

It is worth noting that both rules never change the root node r. The LS is iteratively applied to different roots. This is achieved by starting with a set R containing all n nodes, randomly choosing one to be the root, and removing it from R. Then, the search for minimal e values starts at "level 0". Whenever the solution is changed, the level increases. Glover [11] and Rego [27] found that stopping the search at a relatively low level leads to better performance. After that, another node from R is randomly extracted to be the root for the next iteration.

Glover [11] and Rego [27] defined a "tabu" criterion to prevent generating the same trial solutions repetitively, i.e., no deleted edge should be added back to a S&C structure. This led to the algorithm defined by Rego as `P_SEC` in [27].

Here, we add permutations out of `P_SEC`, refer to it as FSM and improve it in several ways.

First, we propose a new criterion for permitting more moves in order to increase the searchable neighborhood: no previously *deleted* edge can be deleted again (after being re-inserted). This method is denoted as FSM*. We will show that it can produce much better performance not only in terms of speed but also in the quality of the end results. Second, in algorithm version FSM**, we reduce the number of redundant moves by more aggressively limiting the maximum level to $0.45n$ and testing only 15% of the nodes in R as the root node to restart.

Finally, we apply the "soft restart" method defined in [31], where a randomly chosen sub-sequence of the current tour is randomly shuffled. Thus, our `FSM` algorithm is based on three nested loops:

1. The inner-most loop applies rule 1 and rule 2, utilizes the above-mentioned "tabu" criterion, and records the best result during the process.
2. The middle loop iteratively chooses n nodes in the solution as the root and, for each root, applies the inner-most loop.
3. The outer loop takes the best tour created by the middle loop, applies a random change to it (in order to escape potential local optima), and then executes the middle loop again (unless the termination criterion is reached).

2.3 LK Heuristics

Today, the domain of TSP solvers is "ruled" by derivatives of the LK heuristic [23], which can either be considered as a variable k-op heuristic [2, 16, 17] or as an ECM based on a detached stem reference structure [27]. The former perspective is obviously more common.

For ascending values of k, the LK heuristic tries to obtain a shorter tour by replacing k edges. It therefore proceeds as follows to improve a given tour T. Step by step, the algorithm builds two 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. Each edge X_i is chosen such that its start node is the end node of edge Y_{i-1}, while the start node of edge Y_{i+1} is the end node of X_i. By additionally ensuring that the end of Y_k is the start node of X_1, deleting the edges X from T and inserting those from Y will always yield a valid tour. This constitutes a k-opt move, of which the result is accepted if it is shorter than T.

The standard LK heuristic tries all possible choices of X and Y only for $i \leq 2$. While for larger i, no backtracking is allowed. We compare our ECMs with the LK heuristic from [33], where the first improved tour is accepted instead of searching the whole neighborhood, following the suggestion in [16]. This implementation also applies the restart method from [31] used in our `FSM`.

2.4 MNS

While `FSM` and `LK` both investigate potentially very large neighborhoods, the recently proposed MNS applies traditional fixed-k-opt moves. In each iteration,

MNS performs a $\mathcal{O}(n^2)$ scan, which investigates all possible 2-opt and a subset of the possible 3- and 4-opt moves at once. It therefore 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 operation and pair $\{i, j\}$, the gain is computed and all discovered improving moves enter a queue. The access to distance matrix D is minimized by remembering (and updating) the lengths of all n edges in the current tour and avoiding checking redundant moves (swapping the cities at indexes 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. This may invalidate 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 for additional speed-up. If no improving moves can be found anymore, the same "soft restart" method as discussed before is applied [31].

2.5 MAs

EAs are global optimization algorithms inspired by the natural processes of selection and reproduction. They start with a set of λ (usually random) solutions. Among them, the best $\mu \leq \lambda$ solutions are chosen as "parents" of the second generation, which is generated by applying a (unary) mutation or a (binary) crossover operator to the "parents". From then on, the parents are the μ best individuals from the joined set of parents and offspring [$(\mu + \lambda)$-EA].

MAs are EAs hybridized with LS. In our MAs, the LS algorithm is applied to every solution generated by crossover. We use Edge Crossover [32] at a crossover rate of 100%. Edge Crossover is one of the best crossover operators for the TSP. It generates a new solution by picking edges belonging to either of its two parents.

We introduce three different MAs, namely hMA($\mu+\lambda$)ECM, hMA($\mu+\lambda$)LK and hMA($\mu+\lambda$)MNS. The little h at the beginning of the name means that the initialization of these MAs is generated by the Edge-Greedy, Double Minimum Spanning Tree, Savings, Double-Ended Nearest Neighbor and Nearest Neighbor heuristics, as in [31].

2.6 Memetic ACO

The ACO algorithm is inspired by the way ants find and enhance short paths during foraging by using pheromones for communication [6]. PACO [13] is considerably the best-performing ACO variant. Different from standard ACO, which requires storing a pheromone matrix of size in $\mathcal{O}(n^2)$, PACO has linear memory requirements. PACO(k,m) maintains a population of k solutions and the amount of pheromone on an edge of the TSP is proportional to how many solutions this

edge has. For every algorithm iteration, m solutions are created and the "oldest" solution in the entire population is replaced by the best solution in the new generation. Similar to our MAs, our hybrid PACO algorithms are defined in a way that each new solution generated is used as the starting point of a LS procedure whose output then competes to join the population. Such hybrids have performed the best in [31]. Like in our MAs as well as our algorithms from [31], we initialize the first population heuristically.

3 Experiments and Results

3.1 Experimentation with Anytime Algorithms

Most metaheuristics, including EAs, MAs, ACO, as well as all LS methods, are anytime algorithms [4]. Anytime algorithms can provide a best guess of what the optimal solution of a problem could be at *any time during* their run. Experiments for analyzing the behavior of an algorithm over runtime are therefore essential, but are rarely done due to the amount of data they generate and the amount of work required in evaluating the data.

Our *TSP Suite* [31] focuses on investigation of TSP solvers, where data is automatically collected during the evaluation of candidate solutions. Reports showing results based on the data can be automatically generated and freely configured. They contain in-depth descriptions of the experimental procedure and provide several different statistical analyses such as statistical tests comparing the measured runtimes and end results, automated comparisons of the estimated running time (ERT) [15] curves over goal objective values or problem scales, and automated comparisons of empirical cumulative distribution functions (ECDFs) [15, 19, 29]. All of the information is aggregated into human-readable conclusions about the algorithm performance in the form of global rankings.

The *TSP Suite* is the very first framework addressing the issue of measuring runtime. Measuring runtime in CPU seconds 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. To provide a balanced overview of algorithm performance, the *TSP Suite* evaluates runtime using four different measures: CPU time, normalized CPU time, FEs, and the number DE of accesses to the distance matrix D of a TSP.

3.2 Experimental Setup

We conducted our experiments using the symmetric *TSPLib* benchmark cases, for which all optima are known. We can therefore define the quality of a solution as relative error f, the factor by which a solution (tour) is longer than the

optimum. Here, $f = 0$ stands for the optimal solution, and $f = 1$ indicates one that is twice as long. We performed 30 independent runs for each setup on all of the 110 symmetric instances in the *TSPLib* to deal with different hard and easy instances [20], with scales n ranging from 14 to above 85900.

Related work usually focuses on fewer and smaller instances. Rego [27], for example, reported results only for 66 instances with n up to 7397. In [26], 8 instances with n from 48 to 666 were used.

We compared the following algorithms: *1)* FSM, *2)* FSM*, *3)* FSM**, *4)* LK, and *5)* MNS. For each algorithm, seven setups were built: The original (pure) algorithm, three hybrids with PACO and three MAs.

3.3 Pure Algorithm Performance

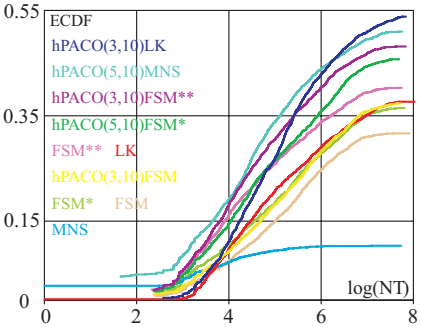
Let us first explore the performance of the three pure versions of FSM (FSM, FSM*, FSM**) and compare them with pure LK and the pure MNS methods. According to the automated ranking provided by the *TSP Suite*, FSM** has the best performance among these algorithms.

In Figure 2, 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 $F_b \leq F_t$ up to a given amount of runtime. Hence, an algorithm is good if its ECDF comes as close to 1 as possible in as soon a manner as possible.

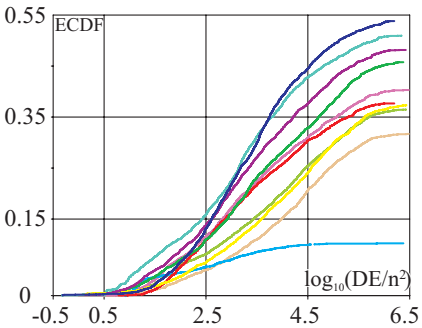
The ECDF of FSM** in Figure 2a, based on the normalized CPU runtime measure *NT*, increases quickly and approaches 0.4 for $F_t = 0$. In other words, a global optimum can been reached in about 40% (of the runs) of all benchmark cases under the given computational budget. Although this does not seem to be very good, FSM** is faster in solving the problems among all pure algorithm settings, as its ECDF curve is always higher than the others. We can furthermore see that the ECDF of FSM* increases a little faster than the one of LK at the beginning, but is eventually overtaken by LK. The performance of FSM is similar to that of FSM* at the beginning, but needs more and more time to solve harder problems. The ECDF of MNS increases slightly faster than the one of FSM** in the beginning, but later slows down and finally reaches a little more than 0.1. For small time budgets, MNS solves more problems than the other methods.

In Figure 2c, we show results with the goal error F_t increased to 0.01, i.e., to investigate the fraction of runs in finding a solution/tour that is no more than 1% longer than the optimum. Again, the ECDF curves intersect. FSM** now solves two thirds of the problems while LK can solve slightly less. For any given time budget, FSM** can solve more instances than any other non-hybrid method. FSM* and FSM are again overtaken by LK as the time goes on.

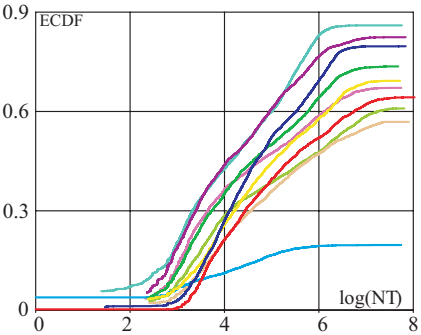
MNS can solve more problems with a small budget of accesses to the distance matrix (Figure 2b). For more *DE*s, we see the same relationship between algorithms as per *NT*. Finally, if we visualize the ECDF under time measure *FE*s in Figure 2d – again for $F_t = 0.01$ – the performance of all three FSM algorithms looks quite similar, except for the later part in the search, during which FSM** can solve significantly more problems than the others.



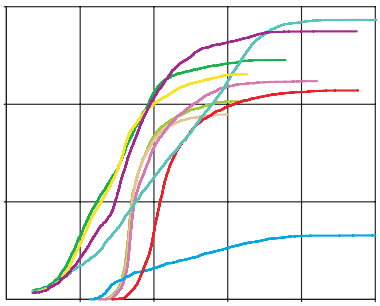
(a) ECDF for NT and $F_t = 0.0$.



(b) ECDF for DE and $F_t = 0.0$.

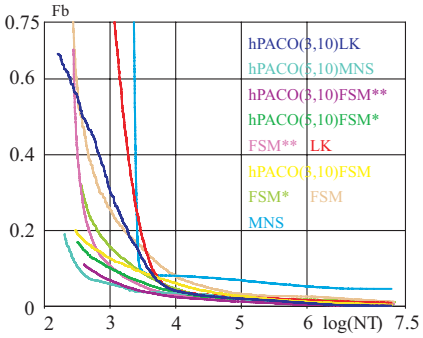


(c) ECDF for NT and $F_t = 0.01$.

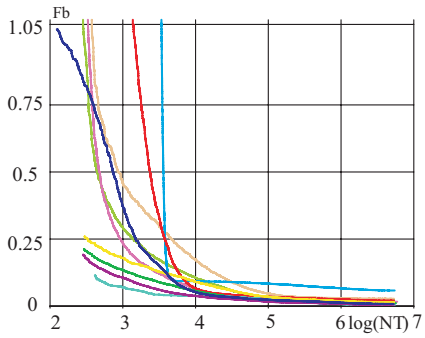


We thereby conclude that FSM** can solve more problems than the other tested pure LS algorithm and that FSM* is better than FSM but worse than LK. In terms of the design criteria discussed in Section 2.2, this means that forbidding edges from being deleted twice is better than forbidding the deletion of added edges (FSM* vs. FSM). Limiting the number of nodes to be tested as the root aggressively further increases the chance of solving the problems faster (FSM** vs. FSM*). We note that a better solution is more probably found at a lower level, in agreement with [12, 27].

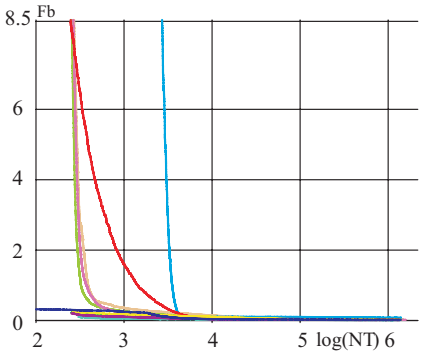
Next, we analyzed the relationship between algorithm performance and problem scale n. We grouped the problems by their scales according to the different powers of two in Figure 3. This figure illustrates the best objective value F_b dis-



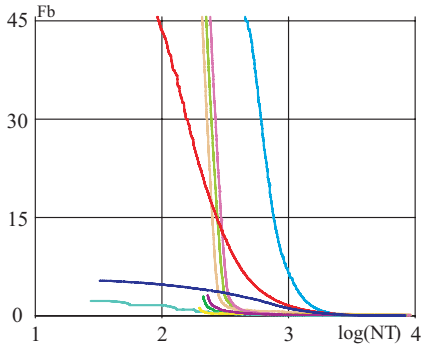
(a) Progress in F_b over NT for $512 \leq n \leq 1023$.



(b) Progress in F_b over NT for $1024 \leq n \leq 2047$.



(c) Progress in F_b over NT for $2048 \leq n \leq 4095$.



(d) Progress in F_b over NT for $16384 \leq n \leq 32767$.

Fig. 3: Progress diagrams for different (log-scaled) time measures and problem scales.

covered by an algorithm over runtime. We see that FSM and LK can find good solutions for instances with n below 1024 quite rapidly. With the increasing of instance scale, the performance of FSM and LK decreases, but they can still find approximate solutions with $F_t \leq 0.05$ for $n < 32768$.

In Figure 3c, we can see that FSM** is always better than LK when $n < 4096$. For a higher scale, LK is better at the beginning but eventually overtaken. FSM* behaves similar to FSM** at the beginning, but is overtaken by both LK and FSM**. FSM is worse than FSM*. Compared to FSM and LK, MNS performs better on small instances but worse for moderate and large-sized ones.

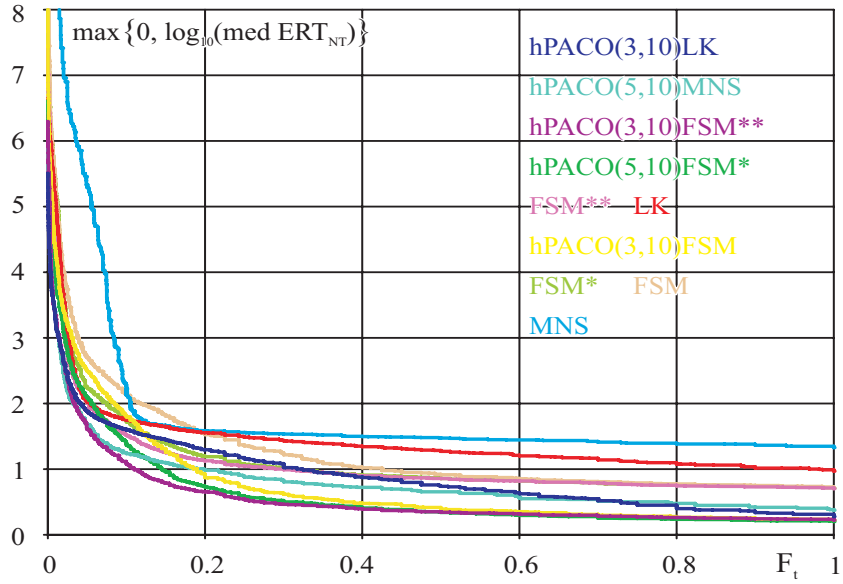


Fig. 4: (Log-scaled) ERT in terms of *DE*.

In Figure 4, we plot the ERT in terms of NT (y-axis) for a given solution quality threshold F_t (x-axis), i.e., the expected normalized runtime it will take for an algorithm to reach F_t. Obviously, the smaller F_t, the higher the expected time. In terms of the ERT measured by counting accesses to the distance matrix as time unit (*DE*s, not illustrated), FSM** is always better than any other pure algorithm in the test. FSM* (FSM) is only faster than LK when an F_t of more than 10% (20%) is acceptable. In terms of NT, the situation changes again and there are two intersections of the ERT curves: MNS performs better for $0.1 \leq F_t \leq 0.5$ while the FSM and LK methods are faster otherwise (but goal errors of 10% and above are of no practical relevance anyway). Thus, if different time measures were used, different observations could be found.

From all the above, we conclude that FSM** is the best LS in our experiments. When the acceptable F_t is big enough, even FSM* and FSM might outperform LK, while pure MNS is the worst.

3.4 Hybrid Algorithm Performance

We also investigated our newly proposed hybridized versions of FSM with EAs (i.e., MAs) and PACO and compared them to similar hybrids of LK and MNS. From the complete ranking generated by the *TSP Suite* over all setups, it can be seen that the different setups of the same corresponding algorithms have relatively similar behaviors, no matter what kind of measure is considered (e.g., hPACO(3,10)MNS and hPACO(5,10)MNS have similar behaviors in terms of ECDF, ERT and progress.). Hybrids with PACO are always better than MAs. We chose the best five setups from the different algorithms, that is, hPACO(5,10)FSM*, hPACO(3,10)FSM, hPACO(3,10)FSM**, hPACO(3,10)LK and hPACO(3,10)MNS, for illustration in the figures. We omitted the MAs as they did not perform as well as the PACO hybrids, even though they were able to outperform the pure algorithms.

From the ECDF curves in Figure 2, we can see some significant improvement of FSM and LK after being hybridized with PACO. hPACO(5,10)FSM*, hPACO(3,10)FSM, hPACO(3,10)FSM** and hPACO(3,10)LK outperform the pure algorithms in terms of both the number of problems they can solve as well as the time they need to solve them. Same observations are made with the MNS hybrids: hPACO(3,10)MNS solves 52% of the problems, which is five times as many as the pure MNS.

hPACO(3,10)MNS is much faster than any other tested algorithms at the beginning, but hPACO(3,10)LK can solve more problems in the end. hPACO(3,10)FSM** is faster than hPACO(3,10)LK at the beginning but eventually overtaken by the latter. Its ECDF is always lower than the one of hPACO(3,10)MNS. hPACO(5,10)FSM* is better than hPACO(3,10)FSM but worse than the other algorithms.

When we set $F_t = 0.01$ (Figure 2c), the hybrid algorithms again outperform the pure ones. Their ECDF curves increase earlier, more rapidly, and finally reach higher end points. hPACO(3,10)FSM** is as fast as hPACO(3,10)MNS and even better at some point, but is finally overtaken as hPACO(3,10)MNS can reach solutions that are one percent longer than the optimum more often. hPACO(3,10)FSM** and hPACO(3,10)MNS are always better than hPACO(3,10)LK. hPACO(5,10)FSM* and hPACO(3,10)FSM can solve more problems than hPACO(3,10)LK early on, but later hPACO(3,10)LK discovers more solutions.

The ERT diagrams in terms of *DE*s for a given F_t share similar trends with those in Figure 4. When F_t is over 0.05, hPACO(3,10)FSM* can be as good as hPACO(3,10)FSM**. For better target tour lengths, hPACO(3,10)FSM** takes over hPACO(3,10)FSM*, and only hPACO(3,10)MNS and hPACO(3,10)LK perform as well as hPACO(3,10)FSM**.

From Figure 3, we again confirm that the FSM hybrids are initially faster than the hybrid LK algorithms but slower than MNS hybrids, although these of LK later on find better-quality solutions.

Regardless of what runtime is available, we confirm that hybrid algorithms are better than pure LS. A more interesting observation, however, is that hybrid MNS methods tend to outperform both hybrid FSM and LK approaches, although pure MNS is clearly much worse than pure FSM (which in turn is better than pure LK). While an almost "additive" effect of hybridization was observed in [31], 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 FSM and 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, therefore provides faster convergence. LK and FSM have been the two *best known* LS approaches for the TSP, with decades of research behind them. Our results, however, make MNS the *best* candidate for hybridization today.

The aggregated algorithm ranking provided by the *TSP Suite* when comparing all setups regarding ECDF, ERT, final results, expected runtime to the optimum, and progress according to different runtime measures is:

hPACO(5,10)MNS, hPACO(3,10)MNS, hPACO(3,25)MNS, hPACO(3,10)FSM**, hPACO(3,25)FSM**, hPACO(5,10)FSM**, hMA(2+4)FSM**, hMA(16+64)FSM**, hMA(16+64)MNS, hMA(2+4)MNS, hMA(2+8)FSM**, hPACO(5,10)LK, hPACO(3,25)LK, hMA(2+8)MNS, hPACO(3,10)LK, hPACO(5,10)FSM*, hPACO(3,25)FSM*, hPACO(3,10)FSM*, hMA(16+64)FSM*, hMA(16+64)LK, hMA(2+8)LK, hMA(2+4)LK, hMA(2+4)FSM*, hMA(2+8)FSM*, hPACO(3,10)FSM, FSM**, hPACO(5,10)FSM, hMA(16+64)FSM, hPACO(3,25)FSM, hMA(2+8)FSM, hMA(2+4)FSM, LK, FSM*, FSM, MNS.

This ranking also reveals that PACO appears to be a better method to hybridize with than an EA. This is exactly the same observation already made in [31].

4 Conclusions and Future Work

In this work, we have introduced a new and improved FSM (FSM**), which can outperform the state-of-the-art of this algorithm family (FSM) as well as the state-of-the-art in LS for the TSP in general (LK). We also introduced new hybrid versions of these algorithms with both EAs and PACO. We compared these hybrids with similar hybrids of LK and the best hybrid algorithms in our *TSP Suite*, hybrid MNS. We conducted a large-scale experiment based on all 110 symmetric instances from the *TSPLib*, performing 30 runs per setting, each limited to 1 hour of runtime. Our experiments have led us to four major conclusions:

1. The new, pure FSM** algorithm works well on both small and large TSP instances and outperforms LK and MNS.

2. The restriction that no edge may be deleted twice from a solution is better than the criterion that deleted edges cannot be added again as used in [12, 27].
3. Another measure to improve performance is restricting the number of levels of the search, as pointed out by Glover [12] and Rego [27]. We additional find that testing fewer root nodes also improves the performance.
4. Based on the same LS algorithm, the best setup of hybrid PACO is always better than any setup of hybrid EAs (MAs). The hybrids of FSM** with PACO are the best variants among other FSM-based algorithms.
5. Although pure FSM and LK is better than MNS, hybrid MNS outperforms hybrid FSM and LK.

The last point is especially interesting and deserves more exploration. We will investigate hybrid EC-LK/FSM/MNS methods, which can use either FSM, LK or MNS as LS. 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 FSM and LK. This could utilize the initial high speed of MNS hybrids while also leverage the better end result quality provided by FSM and LK hybrids. At present, we are also investigating several Tabu Search variants and comparing them with the algorithms presented here.

Acknowledgements 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, and the European Union 7th Framework Program under Grant 247619. The experiments reported in this paper were executed on the supercomputing system in the Supercomputing Center of University of Science and Technology of China.

Bibliography

[1] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton, NJ, USA: Princeton University Press (2007)

[2] Applegate, D.L., Cook, W.J., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. INFORMS Journal on Computing 15(1), 82–92 (2003)

[3] Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.): Handbook of Evolutionary Computation. New York, NY, USA: Oxford University Press (1997)

[4] Boddy, M.S., Dean, T.L.: Solving time-dependent planning problems. Tech. Rep. CS-89-03, Providence, RI, USA: Brown University (1989)

[5] De Jong, K.A.: Evolutionary Computation: A Unified Approach. Cambridge, MA, USA: MIT Press (2006)

[6] Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Milano, Italy: Dipartimento di Elettronica, Politecnico di Milano (1992)

[7] Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization – artificial ants as a computational intelligence technique. IEEE Computational Intelligence Magazine 1(4), 28–39 (2006)

[8] Fogel, D.B.: An evolutionary approach to the traveling salesman problem. Biological Cybernetics 60(2), 139–144 (1988)

[9] Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric tsps by ant colonies. In: Proc. of IEEE Intl. Conf. on Evolutionary Computation. pp. 622–627. Los Alamitos, CA, USA: IEEE, Nagoya, Aichi, Japan: Symposium & Toyoda Auditorium (1996)

[10] Glover, F.: Ejection chains with combinatorial leverage for the traveling salesman problems. Tech. rep., University of Colorado-Boulder (1992)

[11] Glover, F.: New ejection chain and alternating path methods for traveling salesman problems. Computer science and operations research 1992, 449–509 (1992)

[12] Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65(1), 223–253 (1996)

[13] Guntsch, M., Middendorf, M.: Applying population based aco to dynamic optimization problems. In: From Ant Colonies to Artificial Ants – Proc. of the 3rd Intl. Workshop on Ant Colony Optimization (ANTS'02). pp. 111–122. Berlin, Germany: Springer, Brussels, Belgium (2002)

[14] Gutin, G.Z., Punnen, A.P. (eds.): The Traveling Salesman Problem and its Variations. Norwell, MA, USA: Kluwer Academic Publishers (2002)

[15] Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking: Experimental setup. Tech. rep., Orsay, France: Université Paris Sud, INRIA Futurs, TAO (2012)

[16] Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. Tech. rep., Denmark: Roskilde University (1998)

[17] Helsgaun, K.: General k-opt submoves for the lin-kernighan tsp heuristic. Mathematical Programming Computation 1(2-3), 119–163 (2009)
[18] Holland, J.H.: 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)
[19] Hoos, H.H., Stützle, T.: Evaluating las vegas algorithms – pitfalls and remedies. In: Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI'98). pp. 238–245. San Francisco, CA, USA: Morgan Kaufmann, Madison, WI, USA (1998)
[20] Jiang, H., Sun, W., Ren, Z., Lai, X., Piao, Y.: Evolving hard and easy traveling salesman problem instances: A multi-objective approach. In: Simulated Evolution and Learning, pp. 216–227. Springer (2014)
[21] Larrañaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Jour. of Artificial Intelligence Res. 13(2), 129–170 (1999)
[22] Lawler, E.L.G., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Chichester, UK: Wiley (1985)
[23] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research 21(2), 498–516 (1973)
[24] Liu, W., Weise, T., Wu, Y., Chiong, R.: Hybrid Ejection Chain Methods for the Traveling Salesman Problem. In: Gong, M., Pan, L., Song, T., Tang, K., Zhang, X. (eds.) Proceedings of the 10th International Conference on Bio-Inspired Computing – Theories and Applications (BIC-TA'15). Communications in Computer and Information Science, vol. 562, pp. 268–282. Berlin/Heidelberg: Springer-Verlag (Sep 25–28, 2015)
[25] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Germany: Springer (1996)
[26] Pesch, E., Glover, F.: TSP ejection chains. Discrete Applied Mathematics 76(1), 165–181 (1997)
[27] Rego, C.: Relaxed tours and path ejections for the traveling salesman problem. European Journal of Operational Research 106(2), 522–538 (1998)
[28] Rego, C., Gamboa, D., Glover, F., Osterman, C.: Traveling salesman problem heuristics: Leading methods, implementations and latest advances. European Journal of Operational Research 211(3), 427–441 (2011)
[29] Tompkins, D.A.D., Hoos, H.H.: Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. In: Revised Selected Papers from the 7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04). pp. 306–320. Berlin, Germany: Springer, Vancouver, BC, Canada (2004)
[30] Weise, T.: Global Optimization Algorithms – Theory and Application. Germany: it-weise.de (self-published) (2009)
[31] Weise, T., Chiong, R., Tang, K., Lässig, J., Tsutsui, S., Chen, W., Michalewicz, Z., Yao, X.: Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. IEEE Computational Intelligence Magazine 9(3), 40–52 (2014)

[32] Whitley, L.D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesman: The genetic edge recombination operator. In: Proc. of the 3rd Intl. Conf. on Genetic Algorithms. pp. 133–140. San Francisco, CA, USA: Morgan Kaufmann, Fairfax, VA, USA (1989)
[33] Wu, Y., Weise, T., Chiong, R.: Local search for the traveling salesman problem: A comparative study. In: Proc. of 14th IEEE Conf. on Cognitive Informatics & Cognitive Computing. pp. 213–220 (2015)

@inproceedings{LWWC2015HECMFTTSP,
  author   = {Weichen Liu and Thomas Weise and Yuezhong Wu and Raymond Chiong},
  title   = {{Hybrid Ejection Chain Methods for the Traveling Salesman Problem}},
  publisher = {Berlin/Heidelberg: Springer-Verlag},
  booktitle = {Proceedings of the 10th International Conference on Bio-Inspired
               Computing -- Theories and Applications (BIC-TA'15)},
  editor   = {Maoguo Gong and Linqiang Pan and Tao Song and Ke Tang and Xingyi Zhang},
  pages   = {268--282},
  series   = {Communications in Computer and Information Science},
  volume   = {562},
  location = {Hefei, Anhui, China},
  month   = sep # {~25--28,},
  year   = {2015},
  isbn   = {978-3-662-49013-6},
  doi   = {10.1007/978-3-662-49014-3_25},
},