
An Experimental Evaluation of Ejection Chain Algorithms for the Traveling Salesman Problem

Dorabela Gamboa^a, Colin Osterman^b, César Rego^{b*}, Fred Glover^c

^a Escola Superior de Tecnologia e Gestão de Felgueiras, Instituto Politécnico do Porto, Apt. 205, 4610-156, Felgueiras, Portugal. dgamboa@estgf.ipp.pt

^b School of Business Administration, University of Mississippi, University, MS 38677, USA. {costerman; crego@bus.olemiss.edu}

^c Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA. fred.glover@colorado.edu

Latest Revision: February, 2006

Abstract – Ejection chain methods lead the state-of-the-art in local search heuristics for the traveling salesman problem (TSP). The most effective local search approaches primarily originate from the Stem-and-Cycle ejection chain method and the classical Lin-Kernighan procedure, which can be viewed as an instance of an ejection chain method. This paper describes major components of the most effective ejection chain algorithms that are critical for success in solving large scale TSPs. A performance assessment of foremost algorithms is reported based upon an experimental analysis carried out on a standard set of symmetric and asymmetric TSP benchmark problems.

Keywords: traveling salesman problem, ejection chains, local search

* Corresponding author.

1. Introduction

The Traveling Salesman Problem (TSP) consists in finding a minimum distance tour of n cities, starting and ending at the same city and visiting each other city exactly once. In spite of the simplicity of its problem statement, the TSP is exceedingly challenging and is the most studied problem in combinatorial optimization, having inspired well over a thousand publications.

In graph theory, the problem can be defined on a graph $G = (V, A)$, where $V = \{v_1, \dots, v_n\}$ is a set of n vertices (nodes) and $A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ is a set of arcs, together with a non-negative cost (or distance) matrix $C = (c_{ij})$ associated with A . The problem is considered to be symmetric (STSP) if $c_{ij} = c_{ji}$ for all $(v_i, v_j) \in A$, and asymmetric (ATSP) otherwise. Elements of A are often called edges (rather than arcs) in the symmetric case. The version of STSP in which distances satisfy the triangle inequality ($c_{ij} + c_{jk} \geq c_{ik}$) is the most studied special case of the problem. The STSP (ATSP) consists in determining the Hamiltonian cycle (circuit), often simply called a *tour*, of *minimum cost*.

The TSP is a classic NP-hard combinatorial problem, and therefore there is no known polynomial-time algorithm (and unless $P = NP$, none exists) that is able to solve all instances of the problem. Consequently, heuristic algorithms are used to provide solutions that are of high quality but not necessarily optimal. The importance of identifying effective heuristics to solve large-scale TSP problems prompted the “8th DIMACS Implementation Challenge”, organized by Johnson, McGeogh, Glover, and Rego [17] and solely dedicated to TSP algorithms.

In this paper we focus on heuristics based on ejection chain methods because they have proven to dominate other known approaches, solving TSP problems of vastly greater size and difficulty than would have been imagined possible before the advent of recent algorithmic developments. We also describe the state-of-the-art data structures used in the implementation of TSP algorithms, which play a key role in their efficiency.

Although there are several individual publications on ejection chain approaches to TSP, with this paper we intend to provide a new survey that summarizes and compares the best of those approaches that fit into the local search category. Other more general survey publications concerning heuristics for TSP, such as Johnson and McGeoch book chapters [14, 16], are no longer up to date and we include algorithms in our analysis that are not considered in these previous treatments. Furthermore, we introduce and report

computational outcomes for additional algorithms that represent new advances for solving problems in the ATSP class. Finally, we summarize latest developments in data structures that are providing greater efficiencies in solving TSP problems.

The following sections provide a brief survey of the most prominent ejection chain algorithms for the TSP and discuss their salient performance characteristics, together with a summary of computational results that demonstrate the remarkable efficacy of these methods.

2. Symmetric TSP

2.1. Ejection chain based algorithms

Subpath ejection chain methods start from an initial tour and iteratively attempt to improve the current solution, generating moves coordinated by a reference structure. The generation of moves throughout the ejection chain process is based on a set of legitimacy restrictions that determine the set of edges allowed to be used in subsequent steps of constructing the ejection chain. Ejection chains are variable depth methods that generate a sequence of interrelated simple moves to create a compound move.

In the graph theory context, a subpath ejection chain of L levels on graph G consists of a sequence of simple operations, called ejection moves, $\langle e_1, \dots, e_m, \dots, e_L \rangle$, that sequentially transform a subgraph G_m of G into another subgraph G_{m+1} by disconnecting a subpath and reconnecting it with different components. At each level of the chain the subgraph may not represent a feasible solution (usually the reference structure does not correspond to a solution), but it is always possible to obtain a solution to the problem by applying an extra operation called a trial move. Therefore, a neighborhood search ejection chain procedure consists in generating a sequence of moves $\langle e_1, t_1, \dots, e_m, t_m, \dots, e_L, t_L \rangle$, where $\langle e_m, t_m \rangle$ represents the paired ejection and trial moves of level m of the chain. The new solution is obtained by carrying out the compound move $\langle e_1, e_2, \dots, e_m, t_m \rangle$, where the subscript m identifies the chain level that produced the best trial solution. For an extensive description of ejection chain methods we refer the reader to [23].

In this section we summarize the main components of the most effective local search ejection chain algorithms and analyze their performance. These algorithms are chiefly based on the Stem-and-Cycle (S&C) procedure and the Lin-Kernighan (LK) heuristic [19].

The S&C procedure is a specialized approach that generates dynamic alternating paths. The classical Lin-Kernighan approach, by contrast, generates static alternating paths. A theoretical analysis of the differences between the types of paths generated by S&C and LK procedures is provided in Funke, Grünert and Irnich [7].

Johnson and McGeoch Lin-Kernighan (LK-JM)

The Lin-Kernighan neighborhood search is designed as a method to generate k -opt moves (which consist in deleting k edges and inserting k new edges) in a structured manner that provides access to a relevant subset of these moves by an efficient expenditure of computational effort. The approach is based on the fact that any k -opt move can be constructed as a sequence of 2-opt moves [4], and a restricted subset of those move sequences can be produced in a systematic and economic fashion.

The method starts by generating a low order k -opt move (with $k \leq 4$) and then creates a Hamiltonian path by deleting an edge adjacent to the last one added. This completes the first level of the LK process. In succeeding levels each move consists of linking a new edge to the unique degree 1 edge that was not adjacent to the last edge added, followed by deleting the sole edge whose removal will generate another Hamiltonian path.

Additional sophistication of the basic method is provided by a backtracking process that allows restarting with an alternative vertex for insertion or deletion of an edge at level i and proceeding iteratively until reaching level L .

The Lin-Kernighan algorithm implementation analyzed in this paper is from Johnson and McGeoch [15], featured among the lead papers of the “8th DIMACS Implementation Challenge” [17]. The results reported for this implementation use Greedy initial solutions, 20 quadrant-neighbor candidate lists, the don’t-look-bits strategy, and the 2-level tree data structure [6] to represent the tour.

We will indicate the primary algorithms that incorporate one or more of these strategies in their design, including the best algorithms as determined by the 8th DIMACS Implementation Challenge. Algorithms that incorporate more innovative structures and that achieve the highest levels of performance are described in greater detail.

Neto's Lin-Kernighan (LK-N)

This implementation is described in [20]. Its main differences from LK-JM are the incorporation of special cluster compensation routines, the use of a candidate set combining 20 quadrant-neighbors and 20 nearest neighbors, and a bound of 50 moves for the LK searches. It also takes advantage of the don't-look-bits technique and the 2-level tree data structure.

Applegate, Bixby, Chvatal, and Cook Lin-Kernighan (LK-ABCC)

This implementation is part of the Concorde library [1] and is based on [2]. It uses Q-Boruvka starting tours, 12 quadrant-neighbors candidate lists, the don't-look-bits technique, and the 2-level tree data structure. LK-ABCC bounds the LK searches by 50 moves, and the backtracking technique is slightly deeper than that of the LK-JM implementation.

Applegate, Cook and Rohe Lin-Kernighan (LK-ACR)

The implementation of this method is very similar to that of the preceding LK-ABCC approach, but the backtracking strategy is even deeper and broader. The depth of the LK searches, by contrast, is half that of the LK-ABCC approach (25 moves). This implementation is based on the design reported in [2, 3].

Helsgaun's Lin-Kernighan Variant (LK-H)

This implementation, described in [13], modifies several aspects of the original Lin-Kernighan heuristic. The most notable difference is found in the search strategy. The algorithm uses larger (and more complex) search steps than the original procedure. Also, sensitivity analysis is used to direct and restrict the search. The algorithm does not employ backtracking, but uses the don't-look-bits technique and the 2-level tree data structure.

LK-H is based on 5-opt moves restricted by carefully chosen candidate sets. Helsgaun's method for creating candidate sets may be the most valuable contribution of the algorithm. The rule in the original algorithm restricts the inclusion of links in the tour to the five nearest neighbors of a given city. LK-JM includes at least 20 nearest quadrant neighbors. Helsgaun points out that edges selected simply on the basis of length may not have the highest probability of appearing in an optimal solution. Another problem with the original type of candidate set is that the candidate subgraph need not be connected

even when a large fraction of all edges is included. This is the case for geometrical problems in which the point sets exhibit clusters.

Helsgaun therefore develops the concept of an α -nearness measure that is based on sensitivity analysis of minimum spanning 1-trees. This measure undertakes to better reflect the probability that an edge will appear in an optimal solution. It also handles the connectivity problem, since a minimum spanning tree is (by definition) always connected. The key idea, in brief, is to assign a value to each edge based on the length of a minimum 1-tree containing it. A candidate set of edges can then be chosen for each city by selecting edges with the lowest values. The effectiveness of α -nearness in selecting promising edges can be further improved by transforming the graph. For this, a subgradient optimization method is utilized that strives toward obtaining graphs in which minimum 1-trees are close to being tours.

By using the α -measure, the cardinality of the candidate set may generally be small without reducing the algorithm's ability to find short tours. In fact, Helsgaun claims that for his initial set of test problems, the algorithm was able to find optimal tours using as candidate edges the 5 α -nearest edges incident to each node.

Nguyen, Yoshihara, Yamamori and Yasunaga Lin-Kernighan Variant (LK-NYYY)

A short description of this implementation can be found in [17]. This variant starts with a 5-opt move but uses 3-opt moves in the LK searches as opposed to the LK-H approach that uses 5-opt as a basic move. The LK-NYYY also uses don't-look-bits, Greedy starting solutions, and 12 quadrant-neighbor lists, but it uses a data structure with properties similar to segment trees [6]. The results reported from this algorithm were submitted to the DIMACS Challenge after the summary chapter [17] was finished. An extremely significant difference from the Helsgaun variant is that LK-NYYY is able to run instances up to 1,000,000 nodes whereas LK-H only manages instances up to 85,900 nodes and consumes a significant amount of computational time as is evident in Table 4.

Rego, Glover and Gamboa Stem-and-Cycle (SC-RGG)

The SC-RGG algorithm implements an ejection chain method that differs from the LK procedure in several key ways. Most notably the LK approach uses a Hamiltonian path as the reference structure to generate moves throughout the neighborhood construction. This structure is very close to being a valid TSP solution (it only requires adding an edge to link the two degree 1 nodes to obtain a tour). As a result, the structure implicitly limits the different types of moves it can generate. More general ejection chain methods allow a

diversified set of reference structures which are able to generate moves that the classical TSP neighborhood approaches cannot.

The S&C ejection chain method is based on the stem-and-cycle reference structure [10]. The implementation reported here was designed by Rego [22] and subsequently enhanced by Gamboa, Rego and Glover [8, 9]. The S&C reference structure is a spanning sub-graph of G consisting of a path called a stem $ST = (v_t, \dots, v_r)$ connected to a cycle $CY = (v_r, v_{s_1}, \dots, v_{s_2}, v_r)$. A diagram of a Stem-and-Cycle structure is shown in Figure 1. The vertex v_r in common to the stem and the cycle is called the root, and the two vertices of the cycle adjacent to v_r are called subroots. Vertex v_t is called the tip of the stem.

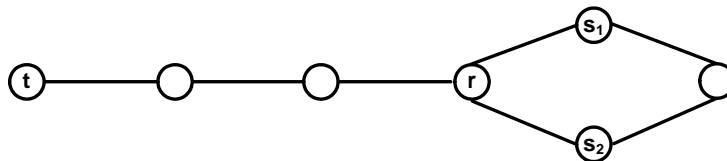


Figure 1 - The S&C reference structure

The method starts by creating the initial S&C reference structure from a TSP tour, by linking two nodes of the tour and removing one of the edges adjacent to one of those nodes. Each ejection move links the tip node to any other node on the graph, except for the one adjacent to the tip. Two different ejection moves are possible depending where in the graph the node to be linked to v_t is placed (in the stem or in the cycle). Trial solutions are obtained by inserting an edge (v_t, v_s) , where v_s is one of the subroots, and deleting edge (v_r, v_s) .

The results reported in this paper improve upon those for the S&C method reported in the DIMACS challenge due to changes outlined in [9] and the addition of a type of don't-look-bits candidate list strategy. Here we present results using Greedy initial solutions, 12 quadrant-neighbor candidate lists concatenated with a list generated by the construction of Delaunay triangulations, and the 2-level tree data structure.

2.2. Comparative analysis of performance

We now evaluate the performance of the heuristic algorithms referenced above using the results submitted to the “8th DIMACS Implementation Challenge” [17] and the updated results for SC-RGG for a comparative analysis. We restrict attention to the evaluation of

the results reported for the algorithms relevant to this paper's main focus. For a complete list of algorithm results and other information related to the generation of the testbed instances, the scale factors to compare running times for different computer systems, and other characteristics of the challenge, we refer the reader to the Challenge web site [17].

The complete Challenge testbed consists of 3 sets of instances: uniformly distributed problems (sizes between 1,000 and 10,000,000 nodes), clustered problems (sizes between 1,000 and 316,228 nodes), and those from the TSP Library [24] with at least 1,000 nodes. In the current study we limited the number of problems to instances up to 3,000,000 nodes.

A benchmark code was provided for Challenge participants that was run on the same machines used to run the competing algorithms of the participants, in order to obtain a more accurate comparison of running times. The tests for the updated version of S&C have been run on the same machine used to run the first S&C version for the DIMACS Challenge, and the same scale factor has been used to normalize the new implementation running times. An exception was made for the 3 million-node problem whose results were obtained on a Dual Intel Xeon, 3.06 GHz with 2GB of memory. A scale factor of 2.89 was used to compute our normalized time for this problem.

Tables 1-4 summarize the results of the aforementioned algorithms. The values presented are averages of solution quality and computational times (in seconds), where instances are grouped by size. This grouping is similar to the one used by Johnson and McGeoch [16] to design the tables of results in their book chapter summarizing the Challenge's submissions. It is important to stress, however, that a number of algorithms and results described here were submitted or updated after the chapter was published. In the solution quality tables, in addition to reporting average percentage excess over the optimal solution or over the Held-and-Karp lower bound, we present the number of best solutions (NBS) found by each algorithm, meaning that for the indicated number of instances the associated algorithm obtained the solution of highest quality. The values in bold indicate the best averages.

We separate the basic LK algorithmic variants and the S&C approach from the other two LK variants since the latter are considerably more sophisticated, placing them in a different category of method by virtue of consuming significantly greater time as they implement more complex and advanced strategies. In particular, the simpler algorithms use 2-opt as a basic move while the more advanced procedures use 3-opt or 5-opt. Basic

LK variants and the S&C method alike determine moves by deleting one edge and inserting another one, completing the 2-exchange with a trial move. The NYYY and Helsingaun variants search for valid 3-exchange and 5-exchange moves. To make this search possible without consuming excessively large amounts of computation time, these procedures use special and highly sophisticated candidate lists as previously noted.

In order to assess the potential effect of using restricted neighborhood search of the type employed by the don't-look-bits strategy considered in the LK implementations, we report results for a first attempt to incorporate this technique in the S&C algorithm. In the tables, SC-RGG⁺ refers to the version of the S&C algorithm that adds restricted neighborhood search to SC-RGG.

From Tables 1 and 2 it is clear that the S&C approach is better than all other implementations for generating high quality solutions. We utilize the notation $x|y$ to signify that the associated algorithm found x better solutions than SC-RGG and y better solutions than SC-RGG⁺ in the corresponding group of problems. To facilitate the analysis of the tables, we replace zeros with dashes and likewise for cases where $x=y$ we use only a single number to avoid repetition. Figure 2 provides a graphical visualization of the results summarized in Table 1. Note that besides achieving better solution quality on average, both S&C variants find significantly larger number of best solutions across all problems and tables. However, it has longer running times, due primarily to our relatively short experience in finding the most effective ways to shortcut the computation required by the don't-look-bits framework.

The graphics in Figure 3 show the effect of the don't-look-bits strategy on the S&C algorithm using the results in Tables 1 and 2. We can see that even a straightforward implementation of the don't-look-bits candidate list strategy produces major reductions in the running times for the S&C algorithm without sacrificing the solution quality. In some cases the quality of the solutions is even better when restricting the neighborhood suggesting that more elaborate implementations of the don't-look-bits strategy can have a dual effect on the performance of the S&C by simultaneously improving the *efficiency* and *effectiveness* of the algorithm. For the uniform distributed problems, the variant of the S&C algorithm that makes use of don't-look-bits (SC-RGG⁺) performs better than its counterpart (SC-RGG), that does not consider such a strategy, in three out of the seven group of problems, while performing comparably on the remaining problems. Also as illustrated in the graphic of Figure 3 depicting the computational times associated with the same testbed, the running times with the don't-look-bits strategy grow sub-linearly

with the problem size while these times are significantly affected in the absence of this strategy as the problem size increases. A similar advantage should be expected for clustered problems, as observed with the LK implementations; hence this topic invites special attention in future developments.

We conjecture that additional improvements can be made in determining more effective neighbor lists that restrict the neighborhood size without omitting arcs that may be critical to perform potentially good moves—not only the best solutions cannot be found if some of the corresponding arcs are not available, but also the search can take much longer to find these solutions when arcs are not made accessible at the appropriate time.

Noticeably, all the aforementioned observations invite the investigation of other more advanced forms of candidate list constructions and strategies such as those that abound in tabu search proposals.

Finally, the consideration of sophisticated techniques like *caching of distances* and other implementation tricks that proved efficient in LK implementations can likewise be incorporated in the S&C algorithm to close the computational gap that still exists between the implementations of the two approaches. (For details on these techniques, we refer to Johnson and McGeoch [15].)

The tables also suggest that LK-JM has some advantages over the clustered instances. From Tables 3 and 4 we can assess the LK-H achieves higher solution quality but with very heavy computational times. This is a serious drawback because the method becomes extremely difficult to use for solving the bigger instances. LK-NYYY obtains reasonably good results in this group of algorithms and is able to report solutions to all instances.

Problem Size/Number of Instances – 25 Uniformly Distributed Problems

Algorithm	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		1000000/1		3000000/1		Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-JM	1,18	1 2	1,27	1 2	2,02	--	2,02	--	1,97	--	1,96	--	1,96	--	1,92	--	1,79	2 4
LK-N	1,17	- 2	1,26	--	1,99	--	1,88	--	1,95	--	1,97	--	1,92	--	1,878	--	1,75	- 2
LK-ABCC	1,47	2 1	1,71	--	2,60	--	2,48	--	2,54	--	2,67	--	2,68	--	2,55	--	2,34	2 1
LK-ACR	1,61	--	2,18	--	2,72	--	2,72	--	2,74	--	2,75	--	2,77	--	2,67	--	2,52	--
SC-RGG	0,79	7	0,95	4	1,68	3	1,61	2	1,65	2	1,86	1	1,91	1	--	--	1,49	20
SC-RGG+	0,93	5	0,98	3	1,55	3	1,66	2	1,72	2	1,84	1	1,90	1	1,875	1	1,56	17+1

Problem Size/Number of Instances – 23 Clustered Problems

Algorithm	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-JM	1,21	6	2,32	4 2	3,41	2	3,72	--	3,63	1 2	3,67	1	2,99	14 13
LK-N	1,97	1	3,55	--	4,76	--	4,42	--	4,78	--	--	--	3,90	1
LK-ABCC	3,22	--	5,58	--	5,70	--	6,38	--	5,31	--	5,45	--	5,27	0
LK-ACR	3,34	--	5,48	--	5,92	--	6,28	--	5,55	--	5,54	--	5,35	0
SC-RGG	1,35	3	2,57	1	3,24	1	3,16	2	3,69	1	3,99	--	3,00	8
SC-RGG+	1,79	3	2,24	3	3,27	1	3,29	2	3,72	--	3,81	--	3,02	9

Problem Size/Number of Instances – 11 TSPLIB Problems

Algorithm	1000/4		3162/3		10000/2		31623/1		100000/1		Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-JM	1,40	--	1,28	--	1,38	--	1,23	--	1,213	--	1,30	--
LK-N	1,43	--	1,44	--	1,34	--	1,49	--	--	--	1,43	--
LK-ABCC	2,56	--	2,41	--	1,86	--	1,65	--	1,208	- 1	1,94	- 1
LK-ACR	3,49	--	2,59	--	3,17	--	2,40	--	2,00	--	2,73	--
SC-RGG	0,52	4	0,60	3	0,91	2	1,02	1	1,17	1	0,84	11
SC-RGG+	0,89	4	0,79	3	0,94	2	1,21	1	1,57	--	1,08	10

Table 1 - Basic LK and S&C – Solution Quality

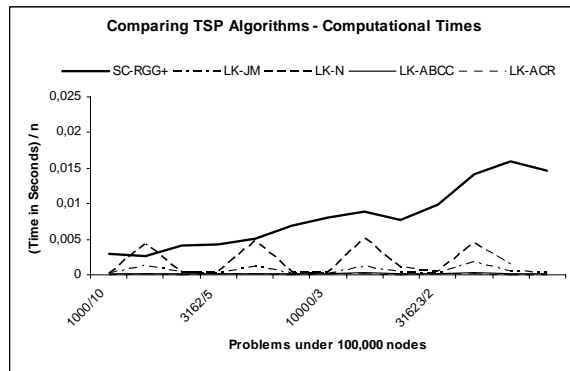
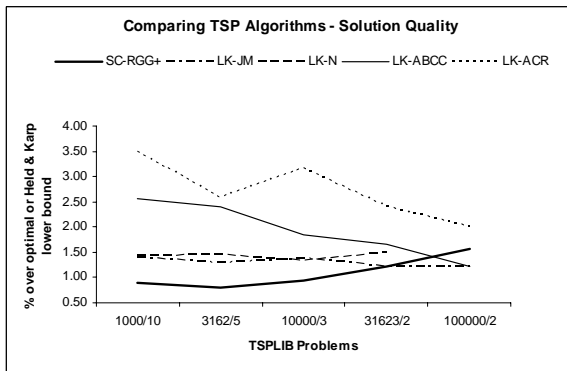
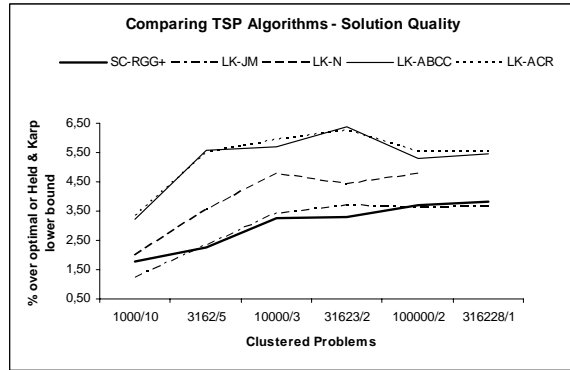
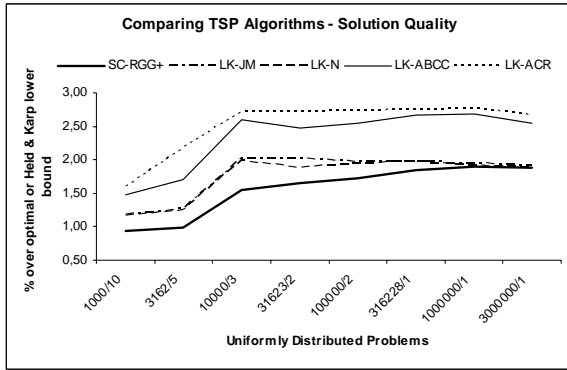


Figure 2 - Basic LK and S&C (values from Tables 1 and 2)

Problem Size/Number of Instances – 25 Uniformly Distributed Problems								
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1	1000000/1	3000000/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU	CPU	CPU
LK-JM	0,16	0,53	1,77	6,81	27,74	108,87	493,42	2049,28
LK-N	0,19	0,87	3,35	14,40	89,58	574,42	3577,74	17660,51
LK-ABCC	0,09	0,34	1,49	5,95	21,43	60,79	307,17	1332,79
LK-ACR	0,07	0,29	0,93	2,95	16,40	76,32	318,10	1289,25
SC-RGG	4,04	19,82	100,92	733,93	5804,09	33239,39	255971,44	--
SC-RGG+	2,95	13,49	80,88	313,11	2872,61	13584,87	69542,57	336304,79

Problem Size/Number of Instances – 23 Clustered Problems						
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU
LK-JM	1,30	3,62	11,99	57,65	211,30	916,91
LK-N	4,35	15,04	51,17	138,59	558,07	--
LK-ABCC	0,20	0,72	2,55	11,04	37,91	107,67
LK-ACR	0,11	0,45	1,40	4,49	24,97	114,19
SC-RGG	4,17	18,41	135,12	956,39	5416,85	60199,97
SC-RGG+	2,58	16,25	89,02	445,76	3818,27	39353,15

Problem Size/Number of Instances – 11 TSPLIB Problems					
	1000/4	3162/3	10000/2	31623/1	100000/1
Algorithm	CPU	CPU	CPU	CPU	CPU
LK-JM	0,29	0,54	3,61	14,57	35,90
LK-N	0,41	1,08	10,26	47,09	--
LK-ABCC	0,10	0,29	1,22	3,48	8,84
LK-ACR	0,08	0,23	0,74	1,74	5,42
SC-RGG	7,59	26,47	134,99	665,85	3253,16
SC-RGG+	4,14	21,67	78,11	505,99	1461,85

Table 2 - Simple LK and S&C – Computation Time

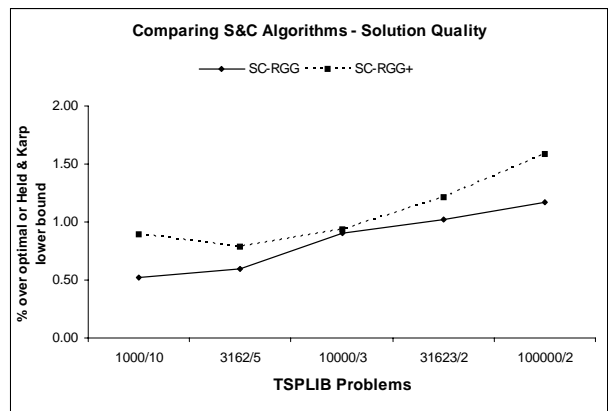
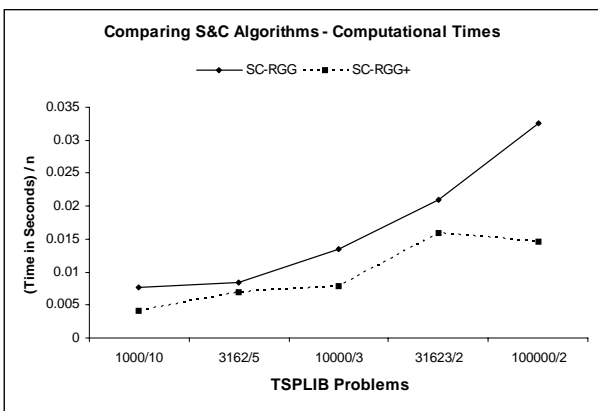
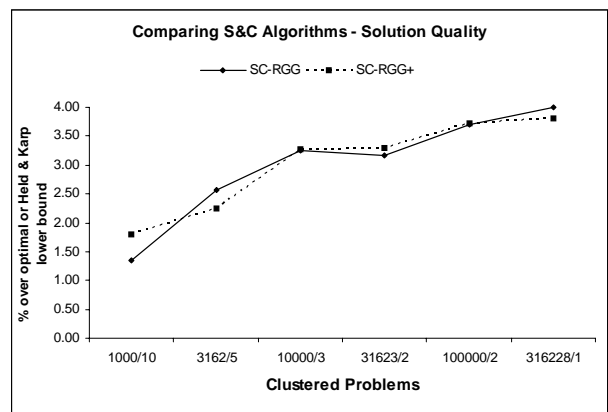
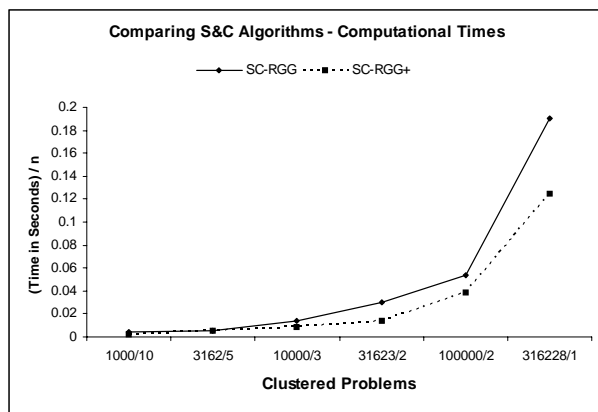
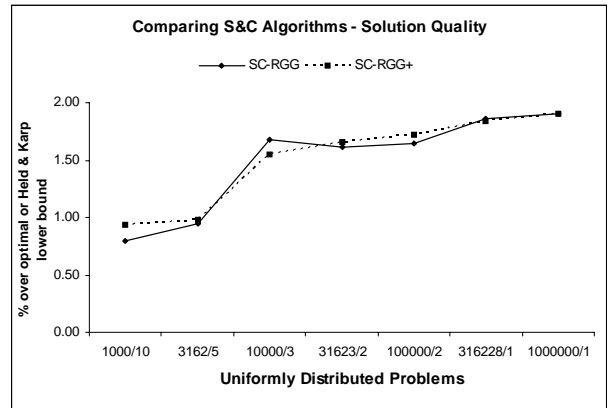
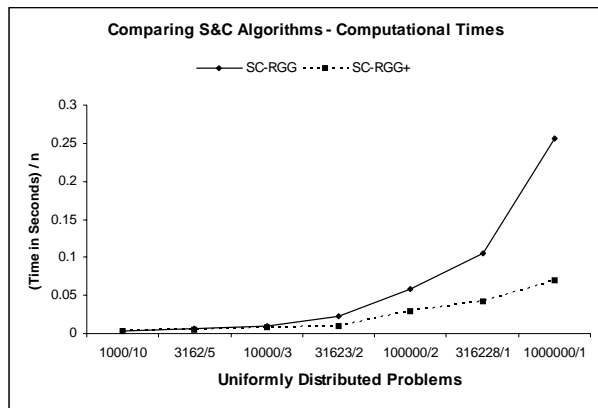


Figure 3 - The Effect of don't-look-bits Strategy on the S&C Algorithm

Problem Size/Number of Instances – 24 Uniformly Distributed Problems																
Algorithm	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		1000000/1		Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-H	0,16	10	0,19	5	0,83	3	0,83	2	--	--	--	--	--	--	0,50	20
LK-NYYY	0,73	--	0,74	--	1,57	--	1,48	--	1,48	2	1,53	1	1,49	1	1,29	4

Problem Size/Number of Instances – 23 Clustered Problems																
Algorithm	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1				Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-H	0,71	8	1,38	4	3,32	1	3,58	1	--	--	--	--	--	--	2,25	14
LK-NYYY	1,22	2	2,18	1	3,08	2	3,45	1	3,51	2	3,49	1	--	--	2,82	9

Problem Size/Number of Instances – 11 TSPLIB Problems																
Algorithm	1000/4		3162/3		10000/2		31623/1		100000/1						Total	
	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK-H	0,24	4	0,15	3	0,24	2	0,46	1	0,85	1	--	--	--	--	0,39	11
LK-NYYY	1,15	--	0,86	--	0,72	--	0,99	--	1,03	--	--	--	--	--	0,95	0

Table 3 - Helsgaun & NYYY – Solution Quality

Problem Size/Number of Instances – 24 Uniformly Distributed Problems							
Algorithm	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1	1000000/1
	CPU	CPU	CPU	CPU	CPU	CPU	CPU
LK-H	5,64	71,49	861,71	7819,27	--	--	--
LK-NYYY	0,16	0,57	1,76	4,97	20,86	84,73	507,62

Problem Size/Number of Instances – 23 Clustered Problems						
Algorithm	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1
	CPU	CPU	CPU	CPU	CPU	CPU
LK-H	6,93	70,28	768,31	12812,46	--	--
LK-NYYY	0,50	1,36	3,96	9,68	38,81	147,20

Problem Size/Number of Instances – 11 TSPLIB Problems					
Algorithm	1000/4	3162/3	10000/2	31623/1	100000/1
	CPU	CPU	CPU	CPU	CPU
LK-H	7,82	73,32	1063,13	7982,09	48173,84
LK-NYYY	0,26	0,66	1,96	5,09	13,06

Table 4 - Helsgaun & NYYY – Computational Time

2.3. Advances in data structures for large STSPs

The problem of data representation is fundamental to the efficiency of search algorithms for the TSP and particularly important for large STSP instances. The nature of these algorithms necessitates the performance of certain basic tour operations involving subpath reversal and traversal. The computational effort that must be devoted to these operations becomes increasingly pronounced with larger problem instances. For example, if the tour is represented as an array (or doubly linked list) of nodes, a subpath reversal takes time $O(n)$, where n is the problem size.

We have recently developed a new data structure—the *k-level satellite tree* [21]—for the purpose of minimizing the contribution of tour management toward the overall runtime cost of a given search.

The *2-level tree* [6] has for many years been considered the most practical choice for representing the tour, retaining that reputation until the recent emergence of the *k-level satellite tree* described herein. A worst-case cost of $O(\sqrt{n})$ for tour operations may be achieved using the 2-level tree representation. The idea is to divide the tour into roughly \sqrt{n} segments, where each segment is maintained as a doubly linked list and the segments are connected in a doubly linked list.

The *k-level satellite tree* takes the segmentation idea a step further: the tour is divided into segments containing roughly $n^{1/k}$ nodes each, and the resulting segments are grouped into *parent segments* containing about $n^{1/k}$ segments each. Ultimately, $k-1$ groupings are performed, giving the tree k levels with at least $n^{1/k}$ parents on the top level. The leveraging effect achieved by this grouping of nodes into segments is the same as that achieved by the 2-level tree, except that we no longer assume that “2” is always the appropriate number of levels.

The 2-level tree representation reduces the time complexity of move operations but pays for it with slightly larger constant costs, also called *overhead*. One might guess that choosing higher values for k (making the tree “taller”) would further reduce complexity while driving up overhead. It turns out that when these costs are balanced, the best value for k increases logarithmically with n , but only approximately, since k must be integer. A related property is that, in most cases, the ideal size of a segment will remain the same as problem size increases. This can be shown algebraically under the assumption, simply

put, that a given algorithm will splice the tree during moves about as often as it will traverse parents. Therefore, the key to choosing k is discovering the ideal segment size. This value, however, varies depending on the design and tuning of a given algorithm, and therefore should be determined experimentally.

Some of the overhead associated with introducing additional levels may be defrayed by utilizing a *satellite* design [21]. A satellite list is similar to a doubly linked list but is symmetric in that an orientation is not inherent. Furthermore, there are no drawbacks in its practical use. The traditional 2-level tree incorporates doubly linked lists. If these are replaced with satellite lists, many of the query operations required in the course of a given algorithm may be performed more quickly than would be possible otherwise. This benefit becomes more pronounced when the tree is expanded to include more than two levels.

In summary, the tour is most efficiently represented with a k -level satellite tree in which k is chosen appropriately. The best value for k can be calculated according to the size of the instance and the ideal segment size, which is unique to each algorithm implementation and must be determined experimentally.

Recent experiments show the k -level satellite tree representation to be far more efficient than its predecessors. Particularly outstanding reductions in algorithm running times occur with large problem instances. When the tree is created with k chosen optimally in comparison to $k=2$, the average running time reduction balloons from a modest 7% for 1,000 node problems to 27% for 10,000 node problems and to 71% for 100,000 node problems. For these tests, a S&C algorithm implemented with the k -level satellite tree was run on Euclidean instances from the DIMACS Challenge [17].

Fortunately, leading ejection chain algorithms for the TSP are similar enough that they may all make use of the same data structures. Consequently, the improvement offered by the k -level satellite tree may be shared as a common advantage in the same way that the 2-level tree has been incorporated in multiple implementations.

3. Asymmetric TSP

3.1. Ejection chain based algorithms

The Kanellakis-Papadimitriou (KP) heuristic [18] (based on the LK procedure) was the only local search ejection chain algorithm for the ATSP submitted to the “8th DIMACS Implementation Challenge” as reported by Johnson and McGeoch in the Challenge

summary chapter [14]. The other two algorithms presented here were not submitted to the Challenge. These are the ATSP version of the S&C algorithm described in the previous section and a new approach for the ATSP using the doubly-rooted stem-and-cycle reference structure [11].

Kanellakis-Papadimitriou Heuristic (KP-JM)

Lin and Kernighan were not concerned with the ATSP when they developed their TSP heuristic in 1973 [19]. LK is based on 2-opt moves which always imply segment reversals that entail exceedingly high computational effort, and hence this method can not be directly applied to the ATSP. A variant of the LK approach presented by Kanellakis and Papadimitriou in 1980 [18] solved this problem by using segment reordering instead of segment reversals (creating and breaking cycles so that the resulting sequence corresponds to a sequence of 3-opt moves). The KP method starts with a variable-depth search based on LK but where the moves performed correspond to k -opt moves for odd values of $k \geq 3$. When the variable-depth search fails to improve the solution, the method searches for an improving 4-opt *double-bridge* move (with no reversals). Then KP returns to variable-depth search and iterates in this manner until neither of the searches improves the tour.

The KP algorithm implementation analyzed in this paper is due to Johnson and McGeoch and described in Cirasella et al. [5]. It takes advantage of the same speedup techniques used in the authors' LK implementation [15], including neighbor lists and the don't-look-bits candidate list strategy. It also uses the dynamic programming approach introduced by Glover [12] to find the best 4-opt move in $O(n^2)$ time.

Rego, Glover, Gamboa Stem-and-Cycle (SC-RGG)

This algorithm is based on the S&C ejection chain algorithm for the STSP by the same authors but ignores moves that generate path reversals. This implementation does not use candidate lists to reduce the neighborhood size, thereby penalizing the computation times as discussed in the following subsection.

Rego, Glover, Gamboa and Osterman Doubly-Rooted S&C (DRSC-RGGO)

The main distinguishing feature of this approach is its use of the doubly-rooted stem-and-cycle reference structure defined in Glover [11], which generalizes the S&C structure by allowing for additional moves on each level of the ejection chain. The doubly rooted structure has two forms: a *bicycle* in which the roots are connected by a single path,

joining two cycles, and a *tricycle* in which the two roots are connected by three paths, thereby generating three cycles (see Figure 4 where r_1 and r_2 indicate the roots).

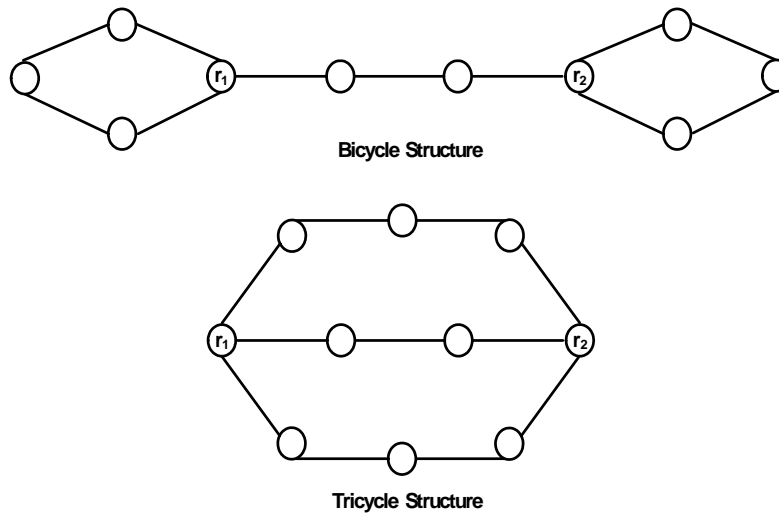


Figure 4 - The Doubly-Rooted reference structure

Ejection moves consist of adding a new edge (s, j) , where s is a subroot and deleting the edge (s, r) resulting in node j as the new root.

In order to assess the effectiveness of the double-rooted S&C neighborhood structure compared to the KP variant, we have adopted for our implementation a similar alternating strategy between the ejection chain search and the 4-opt double-bridge neighborhood. The results reported for this algorithm constitute a preliminary study that is currently being extended. All the implementations use *Nearest Neighbor* starting tours.

3.2. Comparative analysis of performance

Table 5 shows the results reported for the three algorithms on all the TSP Library [24] asymmetric instances with at least 100 nodes. The table shows the percentage deviation above the optimal solution (%), the running times in seconds and average of both values for each algorithm. The best solution for each instance is indicated in bold.

Since our ATSP experiments were not conducted on the same computer considered for the above STSP results (and reported in the Challenge), it is important to explicitly identify the machines used to carry out the tests. The SC-RGG and DRSC-RGGO algorithms were run on an Intel Centrino 1.5GHz processor with 128MB of memory. The results for the

KP-JM algorithm were obtained on the TSP Challenge reference machine, a Silicon Graphics Power Challenge with 31 196 MHz MIPS R10000 processors, 1MB 2nd level caches and 7.6GB of main memory shared by all processors. To be consistent with the analysis reported for the STSP, we provide normalized running times derived from runs of the standard benchmark code available in the Challenge website [17]. We note that the benchmark code used here corresponds to an implementation of the “Hungarian method”, and is different from the Greedy (or Multi-Fragment) geometric benchmark code used above in the normalizations of STSP algorithms. As suggested in [14] such a specialized method for the solution of linear assignment problems is more likely to reflect the pattern of ATSP computations. We encountered relative factors of 1.000, 1.476 and 2.3429 for $n=100$, 316 and 1000, respectively; hence we found 1.6 a reasonable compromise for the actual factors of the two machines.

It is important also to mention that SC-RGG and DRSC-RGGO results were obtained in a single run of the algorithms with fixed parameters. By contrast, results for the KP-JM algorithm are averages over at least 5 runs for each instance. Also, the SC-RGG procedure corresponds to a version of the S&C method created by removing features from its STSP version that do not apply to the ATSP, and no effort has been undertaken to create a specialized S&C variant for asymmetric instances to take advantage of the principles that gave rise to the KP variant of the LK method. Similarly, our 4-opt search used in the current DRSC implementation corresponds to the potentially $O(n^4)$ procedure considered in the original KP algorithm [18], as opposed to Glover’s efficient $O(n^2)$ procedure used in its recent implementations [5] analyzed here.

From Table 5 we can infer that the SC-RGG algorithm obtains competitive results but the DRSC-RGGO approach is clearly more effective in producing high quality solutions. On the 28 instances of the complete testbed, DRSC-RGGO achieves 15 best solutions (and 9 optimal values) as opposed to 4 best solutions (and 3 optimal values) obtained by the KP implementation. The overall percentage deviation average is considerably better, however, for the doubly-rooted S&C approach, although the computational times are higher. These results are displayed in the graphics of Figure 5.

Problem	Algorithm					
	KP-JM		SC-RGG		DRSC-RGGO	
	%	CPU	%	CPU	%	CPU
atex600	4,25	3,38	6,97	98,30	3,54	1440,80
big702	2,10	6,04	3,54	167,34	1,58	692,14
Code198	0,00	0,54	0,00	1,87	0,00	0,10
Code253	0,10	1,09	0,35	3,30	0,00	41,02
dc112	0,39	15,47	0,91	1,33	0,14	26,35
dc126	0,65	22,69	1,68	2,19	0,20	76,14
dc134	0,57	13,43	0,80	2,40	0,21	32,90
dc176	0,67	20,48	2,39	1,95	0,19	112,77
dc188	0,59	12,98	1,19	4,58	0,22	156,43
dc563	0,79	111,95	2,47	46,53	0,93	280,48
dc849	0,62	114,80	0,63	103,30	0,63	214,72
dc895	0,60	144,43	2,18	217,78	0,58	3070,08
dc932	0,26	119,17	1,23	190,35	0,40	2423,55
ftv100	3,11	0,40	1,45	1,12	0,00	7,74
ftv110	4,04	0,40	1,28	1,74	0,31	24,58

Problem	Algorithm					
	KP-JM		SC-RGG		DRSC-RGGO	
	%	CPU	%	CPU	%	CPU
ftv120	3,12	0,50	1,80	2,03	0,92	18,27
ftv130	2,16	0,60	2,90	2,02	0,26	17,17
ftv140	3,15	0,60	2,23	2,48	0,25	68,43
ftv150	4,43	0,70	2,68	2,42	1,80	18,88
ftv160	5,89	0,70	5,63	3,65	0,00	33,81
ftv170	4,44	0,90	3,59	4,35	0,11	171,02
rbg323	0,78	3,71	1,06	26,90	0,08	64,40
rbg358	1,50	3,33	3,44	46,27	0,00	81,46
rbg403	0,22	9,00	0,28	69,94	0,00	33,63
rbg443	0,11	11,74	1,10	107,31	0,00	7,92
td100.1	0,00	0,20	0,09	2,06	0,00	2,30
td1000.20	0,01	7,29	0,06	1106,91	0,10	4373,92
td316.10	0,00	3,87	0,17	52,80	0,00	84,88
Average	1,59	22,51	1,86	81,19	0,44	484,85

Table 5 - Solution Quality & Computational Time

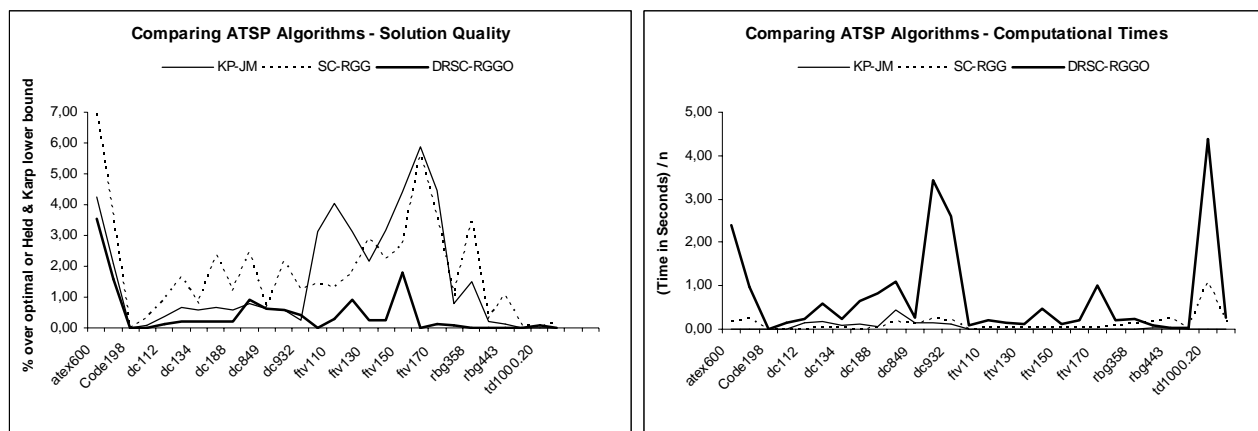


Figure 5 - Comparison of ATSP Algorithms (values from Table 5)

4. Concluding Remarks

In this paper we describe and compare the most effective and efficient local search ejection chain algorithms for the TSP. These algorithms concern variants of the Lin-Kernighan (LK) approach and two variants of the stem-and-cycle (S&C) ejection chain method. We find that the S&C approaches clearly outperform the basic LK implementations.

For symmetric instances, the S&C approach finds better solutions than all (four) of the leading LK variants for about 70% of the problems tested. Conspicuously, the 70% advantage of the S&C approach refers to a comparison with the most effective variant of the LK procedure. The second best variant of this approach is dominated by the S&C approach in approximately 97% of the problems. Some other variants failed to find even a single solution better than the S&C approach over all 59 problems tested.

Similar success was achieved by our doubly-rooted S&C variant applied to the asymmetric setting of the problem. Tests on 28 standard instances revealed 15 best solutions (and 9 optimal values) for our doubly-rooted S&C algorithm as opposed to 4 best solutions (and 3 optimal values) obtained by a specialized LK variant for these asymmetric instances.

We conjecture that gains in performance from the ejection chain methods are accomplished by their ability to use k -opt moves for $k \geq 4$ that are not accessible to the LK approaches. We anticipate that future gains will result by introducing more effective candidate lists that narrow the neighborhood size without causing solution quality to deteriorate. The ejection chain methods not only perform better than the local search TSP algorithms based on the LK framework, but also give the overall best solutions when the local search algorithms described here are used as engines for iterated local search heuristics.

References

- [1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Concorde: A code for solving Traveling Salesman Problems", 1999, <http://www.math.princeton.edu/tsp/concorde.html>.
- [2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Finding Tours in TSP", Research Institut for Discrete Mathematics, Universitat Bonn, Bonn, Germany, 99885, 1999.
- [3] D. Applegate, W. Cook, and A. Rohe, "Chained Lin-Kernighan for Large Traveling Salesman Problems", *INFORMS Journal on Computing*, vol. 15, pp. 82-92, 2003.
- [4] N. Christofides and S. Eilon, "Algorithms for Large-Scale Traveling Salesman Problems", *Operations Research Quarterly*, vol. 23, pp. 511-518, 1972.
- [5] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang, "The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators and Tests", in Proceedings of the Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, pp. 32-59, 2001.
- [6] M. L. Fredman, D. S. Johnson, L. A. McGeoch, and G. Ostheimer, "Data Structures for Traveling Salesman", *Journal of Algorithms*, vol. 18, pp. 432-479, 1995.
- [7] B. Funke, T. Grünert, and S. Irnich, "A Note on Single Alternating Cycle Neighborhoods for the TSP", *Journal of Heuristics*, vol. 11, pp. 135-146, 2005.
- [8] D. Gamboa, C. Rego, and F. Glover, "Data Structures and Ejection Chains for Solving Large-Scale Traveling Salesman Problems", *European Journal of Operational Research*, vol. 160, pp. 154-171, 2005.
- [9] D. Gamboa, C. Rego, and F. Glover, "Implementation Analysis of Efficient Heuristic Algorithms for the Traveling Salesman Problem", *Computers and Operations Research*, vol. 33, pp. 1161-1179, 2006.
- [10] F. Glover, "New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems", *Computer Science and Operations Research*, pp. 449-509, 1992.
- [11] F. Glover, "Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems", *Discrete Applied Mathematics*, vol. 65, pp. 223-253, 1996.
- [12] F. Glover, "Finding a Best Traveling Salesman 4-Opt Move in the Same Time as a Best 2-Opt Move", *Journal of Heuristics*, vol. 2, pp. 169-179, 1996.
- [13] K. Helsgaun, "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic", *European Journal of Operational Research*, vol. 126, pp. 106-130, 2000.
- [14] D. S. Johnson, G. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovitch, "Experimental Analysis of Heuristics for the ATSP", in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. Punnen, Eds. Boston: Kluwer Academic Publishers, pp. 445-487, 2002.
- [15] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization", in *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds.: John Wiley and Sons, Ltd., pp. 215-310, 1997.
- [16] D. S. Johnson and L. A. McGeoch, "Experimental Analysis of Heuristics for the STSP", in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. Punnen, Eds. Boston: Kluwer Academic Publishers, pp. 369-443, 2002.
- [17] D. S. Johnson, L. A. McGeoch, F. Glover, and C. Rego, "8th DIMACS Implementation Challenge: The Traveling Salesman Problem", 2000, <http://www.research.att.com/~dsj/chtsp/>.
- [18] P. C. Kanellakis and C. H. Papadimitriou, "Local search for the asymmetric traveling salesman problem", *Operations Research*, vol. 28, pp. 1086-1099, 1980.
- [19] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem", *Operations Research*, vol. 21, pp. 498-516, 1973.
- [20] D. Neto, *Efficient Cluster Compensation for Lin-Kernighan Heuristics*: Department of Computer Science, University of Toronto, 1999.
- [21] C. Osterman and C. Rego, "The Satellite List and New Data Structures for Symmetric Traveling Salesman Problems", University of Mississippi, HCES-03-06, 2004.
- [22] C. Rego, "Relaxed Tours and Path Ejections for the Traveling Salesman Problem", *European Journal of Operational Research*, vol. 106, pp. 522-538, 1998.
- [23] C. Rego and F. Glover, "Local Search and Metaheuristics", in *The Traveling Salesman Problem and Its Variations*, G. Gutin and A. Punnen, Eds. Dordrecht: Kluwer Academic Publishers, pp. 309-368, 2002.
- [24] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library", *ORSA Journal on Computing*, vol. 3, pp. 376-384, 1991.