Using POPMUSIC for Candidate Set Generation in the Lin-Kernighan-Helsgaun TSP Solver

Keld Helsgaun
E-mail: keld@ruc.dk

Department of Computer Science
Roskilde University
DK-4000 Roskilde, Denmark
July 2018

Abstract

This report describes an enhancement of the Lin-Kernighan-Helsgaun TSP solver (LKH) for fast generation of candidate sets for very-large scale traveling salesman problems. Its implementation is based on a metaheuristic called POPMUSIC. The enhancement makes it possible to generate high-quality candidate sets in almost linear time, even for non-geometric instances.

1. Introduction

A key point for treating large TSP instances is to consider only a subset of edges connecting the cities. For this purpose, it is essential to build a network with an extremely low density, typically by keeping only a few connections (candidate edges) for each city. In the case of geometric problems, several techniques have been proposed for generating an adequate network. For 2-dimensional Euclidean instances, a Delaunay triangulation can be built in $O(n \log(n))$ time, where n is the number of cities in the problem. When the cities are specified with coordinates in K dimensions, another technique is to build a k-d tree (in $O(Kn \log(n))$ time) and to keep only few of the nearest cities in each of the geometric quadrants around each city. Both techniques, which ensure a connected and sparse network, are implemented in the Lin-Kernighan-Helsgaun TSP solver, LKH [1][2].

For non-geometric (as well as for geometric) instances, LKH provides an implementation of a technique based on minimum-spanning 1-trees. For each of the possible $O(n^2)$ edges, a value called α is computed. Given the cost of a minimum 1-tree, the α-value of an edge is the increase of this cost when a minimum 1-tree is required to contain the edge. The α-values provide a good estimate of the edges' chances of belonging to an optimum tour. Node penalties found by subgradient optimization are used to improve this estimate. A sparse and connected network is ensured by selecting the k α-nearest neighbors to each city where k is a small constant. In general, using α-nearness for specifying the candidate set is much better than using ordinary nearest neighbors. Both techniques require quadratic computational effort, but usually, the α-nearness based candidate set may be smaller, without degradation of the solution quality.

Candidate set generation based on 1-trees and α-nearness is default in LKH. It is acceptably fast for instances of up to 100,000 cites. However, for larger instances its quadratic time results in unacceptable execution times. A subquadratic algorithm is needed. It turns out that POPMUSIC fulfils the need.

POPMUSIC (Partial OPtimization Metaheuristic Under Special Intensification Conditions) is a template for tackling large problem instances. This metaheuristic has been shown to be very efficient for various hard combinatorial problems such as *p*-median, sum of squares clustering, vehicle routing, map labelling and location routing. The basic idea of POPMUSIC is to locally optimize sub-parts of a solution, once a solution of the problem is available. These local optimizations are repeated until no improvements are found.

The POPMUSIC template may be used on TSP as follows. Given an initial tour, optimize locally sub-paths of *r* consecutive cities on the tour [3]. This definition allows to easily identify a sub-part of a solution. Moreover, an optimized sub-path can be easily replaced in the current tour.

In the following it will be shown how to obtain reasonably good initial tours and improve them with POPMUSIC in almost linear time. The union of the edges of a specified number of these improved tours constitutes a candidate set.

2. POPMUSIC for LKH

The code below sketches in C-style notation how POPMUSIC is used in LKH for generating a candidate set.

```
 1 for (s = 1; s <= POPMUSIC_SOLUTIONS; s++) {
 2     for (i = 0; i < n; i++)
 3         solution[i] = i;
 4     shuffle(n, solution);
 5     solution[n] = solution[0];
 6     build_path(n, solution, POPMUSIC_SAMPLE_SIZE);
 7     fast_POPMUSIC(n, solution, pow(POPMUSIC_SAMPLE_SIZE, 2));
 8     add_to_candidate_set(n, solution);
 9 }
10 trim_candidate_set(n, MAX_CANDIDATES);
```

Lines 2-4 generates a random tour for an *n*-city instance. An initial tour is built in line 6, and improved by POPMUSIC in line 7. The *n* edges of this improved tour is then added to an initially empty candidate set in line 8. This process of generating solution tours and adding their edges to the candidate set is repeated a specified number of times (POPMUSIC_SOLUTIONS). Finally, in line 10, the candidate set is trimmed so that a specified maximum number (MAX_CANDIDATES) of candidate edges emanates from each city.

POPMUSIC_SOLUTION, POPMUSIC_SAMPLE_SIZE and MAX_CANDIDATES are parameters to LKH with default values 40, 10 and 5, respectively.

3. Building an Initial Tour

An initial tour is built by the function `build_path`.

```
 1 void build_path(int n, int *path, int sample_size) {
 2     if (n <= pow(sample_size, 2))
 3         optimize_path(n, path);
 4     else {
 5         S = select_sample(n, path, sample_size);
 6         S[sample_size + 1] = S[0];
 7         optimize_path(sample_size + 1, S);
 8         for (city = 0; city <= n; city++)
 9             if (!belongs(city, S, sample_size))
10                 insert(city, S, sample_size);
11         for (cluster = 0; cluster < sample_size; cluster++)
12             build_path(size(S[cluster]) + 2, &S[cluster] - 1);
13     }
14 }
```

The function takes as input an array `path[0..n]` of cities and builds recursively a path from `path[0]` to `path[n]`. In line 5 a random sample, `S`, of `sample_size` cities from `path[0..n]` is selected. An attempt is made in line 7 to find an optimal tour for the sample. Lines 8-10 insert any non-sample city after its closest sample city, thereby organizing `path[0..n]` as a sequence of clusters. Lines 11-12 build recursively a path for each of these clusters.

If `n` is less than or equal to `sample_size` squared, an attempt is made to optimize the path (lines 2-3).

Path optimization is done using the 3-opt heuristic. Few nearest neighbor candidates (default is 5), “positive gain criterion” and “don’t look bits” are used for speeding up the heuristic.

4. Improving an Initial Tour

An initial tour is improved using the function `fast_POPMUSIC`.

```
 1 void fast_POPMUSIC(int n, int *path, int r) {
 2     for (scan = 1; scan <= 2; scan++)
 3         if (scan == 2) {
 4             circular_right_shift(n, path, r / 2);
 5         for (i = 0; i < n / r; i++)
 6             optimize_path(r, path + r * i);
 7         if (n % r != 0)
 8             optimize_path(r, path + n - r);
 9     }
10 }
```

The function performs two scans of the given closed path (tour). In each scan, it locally optimizes non-overlapping sub-paths of `r` consecutive cities on `path[0..n]` (lines 5-6). A possible remaining portion of the path is optimized in line 8. A circular right shift of `path` with `r`/2 positions before the second scan (line 4) is used in order to optimize sub-paths involving `r`/2 cities for each of two adjacent sub-paths in the first scan.

4. Reducing the POPMUSIC Candidate Set

A POPMUSIC crated candidate graph is usually quite sparse, even when based on a large number of POPMUSIC solutions. However, for better performance of LKH, the candidate set may be trimmed. The function shown below reduces the POPMUSIC candidate set such that each node has a specified maximum number of emanating candidate edges, `max_candidates` (default is 5).

```
1 void trim_candidate_set(int n, int max_candidates) {
2     subgradient_optimization();
3     compute_alpha_values();
4     for (city = 0; city < n; city)
5         eliminate_candidates(city, max_candidates);
6 }
```

The function attempts to eliminate candidate edges of low prospect of belonging to an optimum tour. The edges' chances of belonging to an optimum tour are estimated using the α-nearness measure. For each candidate edge its α-value is computed (line 3) based on node penalties found by subgradient optimization (line 2). Note that the sparseness of the candidate graph enables this computation to be performed quickly (in $O(n \log n)$ time). Lines 4-5 eliminate, for each city, those emanating candidate edges that are not among the `max_candidates` edges with the smallest α-values.

6. Experimental Evaluation

The incorporation of POPMISIC into LKH has been evaluated on an iMac with an 3.6 GHz Intel Core i7 CPU and 16 GB of RAM running macOS High Sierra operating system. The evaluation is divided into four parts. First, the time efficiency and quality of POPMUSIC solutions are evaluated. Then, the effect of reducing the POPMUSIC candidate set is examined. Next, the overall performance is illustrated using some large-scale instances. Finally, POPMUSIC is compared with two of LKH's built-in candidate set generation algorithms: ALPHA and DELAUNAY.

6.1 Efficiency and Quality of POPMUSIC Solutions

The time usage for generating POPMUSIC solutions as well as their quality have been evaluated by means of Euclidean instances of the DIMACS TSP Challenge, which are instances consisting of uniformly distributed points in a square (*E-instances*) and clustered points in a square (*C-instances*). The advantage of using these instances is that optimal or high-quality solutions are known (found by LKH).

Table 6.1.1 reports the time usage in seconds for generating 50 POPMUSIC solutions, the average percentage excess over the best known tour length, the average node degree in the candidate graph, and the number of edges in the best known solution tour missing from the candidate set.

Instance	*n*	Time (s)	Gap (%)	Degree	Missing
E10k.0	10000	2.3	11.3	7.0	2
E31k.0	31623	7.6	11.9	7.2	2
E100k.0	100000	26.1	12.1	7.2	14
E316k.0	316228	93.6	12.2	7.3	39
E1M.0	1000000	308.8	12.3	7.3	97
E3M.0	3162278	1064.9	12.4	7.3	348
E10M.0	10000000	4012.2	12.4	7.3	1115
C10k.0	10000	2.2	18.4	6.4	1
C31k.0	31623	7.9	19.4	6.4	6
C100k.0	10000	26.8	20.0	6.4	14
C316k.0	3162278	92.1	20.5	6.5	78

Table 6.1.1 Results for generating 50 POPMUSIC solutions (default parameter values)

The time usage for the E-instances is plotted in Figure 6.1.1. As can be seen, the growth is almost linear ($O(n^{1.06})$).

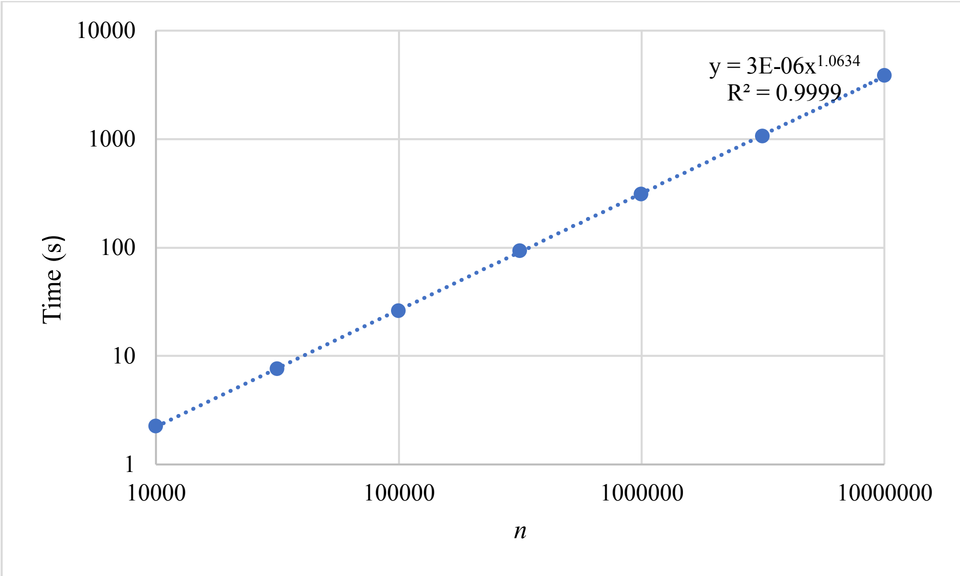

Figure 6.1.1 Computational time for generating 50 POPMUSIC solutions (E-instances)

Although the quality of each POPMUSIC solution tour is modest (10-20%), it turns out that the union of relatively few of them includes all edges of the best known solution. This powerful property is documented in Table 6.1.2, which gives the number of POPMUSIC solutions and the time used for reaching a state where all edges of the best known tour are included in the union of solutions. The last column contains the average node degree for the resulting candidate graph.

Instance	Solutions	Time (s)	Degree
E10k.0	54	2.9	7.3
E31k.0	97	17.1	8.5
E100k.0	90	53.8	8.4
E316k.0	136	382.6	9.3
E1M.0	182	925.2	12.3
E3M.0	187	4406.4	12.4
E10M.0	289	20950.6	11.5
C10k.0	63	3.4	6.7
C31k.0	74	13.5	7.0
C100k.0	187	114.2	8.9
C316k.0	164	344.9	8.7

Table 6.1.2 Results for termination when all edges of a best known tour are included

The number of missing best tour edges as a function of generated POPMUSIC solutions for E10k.0 is depicted in Figure 6.1.3.

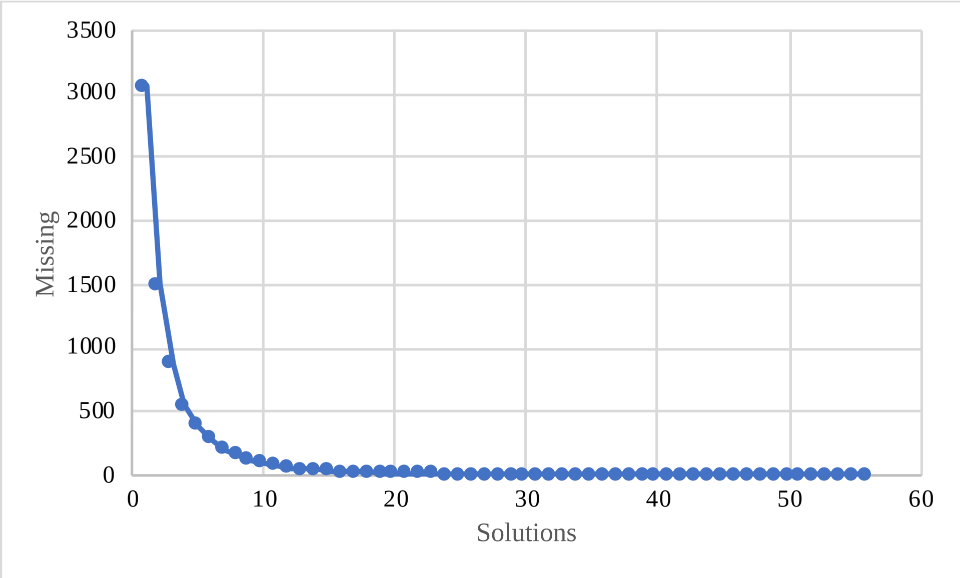

Figure 6.1.3 Missing best tour edges for E10k.0 as a function of POPMUSIC solutions

LKH with POPMUSIC contains four parameters that can be used to govern the candidate generation process. They are:

POPMUSIC_SOLUTIONS:
Number of POPMUSIC solutions to be generated (default: 40)

POPMUSIC_SAMPLE_SIZE:
The sample size (default: 10)

POPMUSIC_TRIALS:
Number of trials used in iterated 3-opt (default: 1).
If the value is zero, the number of trials is the size of the sub-path to be optimized.

POPMUSIC_MAX_NEIGHBORS:
Maximum number of nearest neighbors used in 3-opt (default: 5)

If one is willing to spend more computing time, higher quality of each of the POPMUSIC may be achieved by choosing higher values than their defaults for the last three parameters. Table 6.1.3 shows the results for the parameter settings

CANDIDATE_SET_TYPE = POPMUSIC
POPMUSIC_SOLUTIONS = 1
POPMUSIC_SAMPLE_SIZE = 50
POPMUSIC_TRIALS = 1000
POPMUSIC_MAX_NEIGHBORS = 20

Instance	Gap (%)	Time (s)
E10k.0	1.6	2.1
E31k.0	1.6	5.6
E100k.0	1.9	24.5
E316k.0	2.1	81.3
E1M.0	2.0	223.9
E3M.0	2.0	737.8
E10M.0	2.2	2746.6
C10k.0	3.0	3.3
C31k.0	5.4	8.8
C100k.0	5.3	36.4
C316k.0	5.4	118.8

Table 6.1.3 Results for generating one POPMUSIC solution (non-default parameter values)

However, it should be noted that better POPMUSIC solutions not necessarily lead to better candidate sets. Diversity among solutions appears to be more important than getting as good solutions as possible.

6.2 Effect of Candidate Set Reduction

To speed up the Lin-Kernighan search, the POPMUSIC generated candidate set may be reduced using the α-nearness measure. Good α-values for the edges are found by subgradient optimization. Note that no new edges are added; for each city, its emanating POPMUSIC generated candidate edges are just sorted according to their α-values, and the first MAX_CANDIDATES (default: 5) edges are selected.

Table 6.2.1 reports the experimental results for 6 instances, where the T-instances are uniformly distributed toroidal instances in 2D. The following non-default LKH parameter settings were used:

CANDIDATE_SET_TYPE = POPMUSIC
INITIAL_PERIOD = 100
MAX_TRIALS = 1000

As can be seen, LKH is able to find high-quality solutions for these instances, even though the reduction causes elimination of some of the edges in the best known solution tours.

	E10k.0	C10k.0	T10k.0	E100k.0	C100k.0	T100k.0
POPMUSIC Time (s)	2.3	2.2	2.5	26.1	26.8	28.8
Reduction Time (s)	1.4	1.1	0.8	29.6	24.6	25.1
Lin-Kernighan Time (s)	105.2	236.3	165.8	3830.9	2461.3	4491.1
POPMUSIC Degree	7.0	6.4	7.3	7.2	6.4	7.3
Reduction Degree	4.9	4.8	4.9	4.9	4.8	4.9
POPMUSIC Missing	2	1	2	14	14	13
Reduction Missing	11	102	11	102	167	112
Lin-Kernighan Gap (%)	0.011	0.045	0.022	0.041	0.208	0.062

Table 6.2.1 Effect of candidate set reduction on tour quality (with subgradient optimization)

The reduction may be performed without subgradient optimization (by setting SUBGRADIENT = NO or INITIAL_PERIOD = 0). However, this usually decreases the performance of LKH (see Table 6.2.2).

	E10k.0	C10k.0	T10k.0	E100k.0	C100k.0	T100k.0
Reduction Time (s)	0.2	0.2	0.2	0.3	0.3	0.3
Lin-Kernighan Time (s)	127.8	200.2	164.6	5211.8	4658.1	6380.4
POPMUSIC Missing	2	1	2	14	14	13
Reduction Missing	55	58	63	596	547	667
Lin-Kernighan Gap (%)	0.014	0.052	0.037	0.092	0.200	0.121

Table 6.2.2 Effect of candidate set reduction on tour quality (without subgradient optimization)

The Lin-Kernighan search in LKH uses distances transformed by the node penalties found by subgradient optimization. This has turned out to be advantageous. It is presumably due to a "smoothing" effect upon the original instance. Even a few steps of subgradient optimization often have a positive effect.

Figure 6.2.3 depicts the total time used for generating candidate sets for the E-instance (with candidate set reduction). As can be seen, the growth is almost linear ($O(n^{1.15})$).

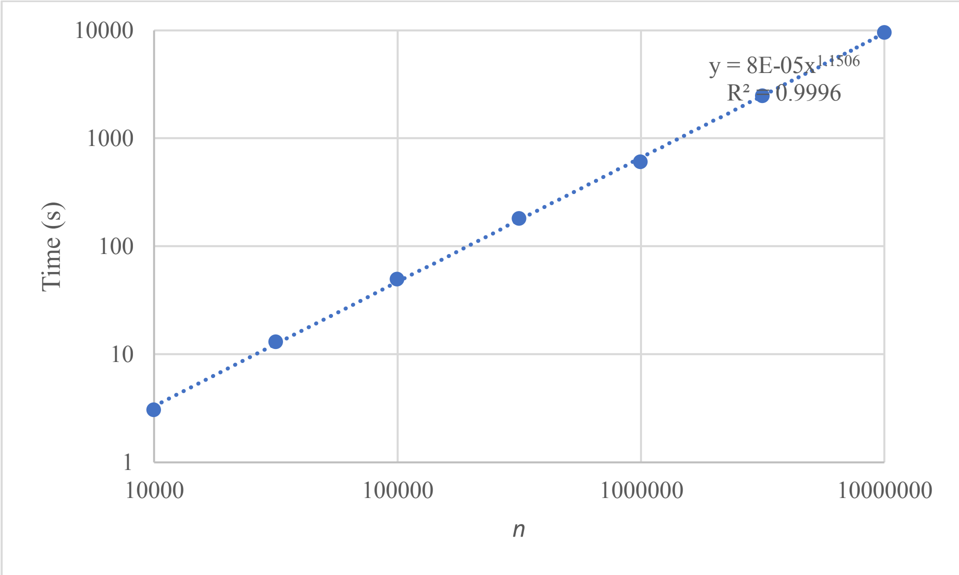

Figure 6.2.3 Total time for generating candidate sets for E-instances

6.3 Performance on some Large-Scale Instances

The performance of POPMUSIC has been evaluated on 5 benchmark instances with about 100,000 cites. Optima are known for two of the instances: pla85900 and star109339 (both found by LKH). High-quality (probably optimal) solutions are known for the other three.

Table 6.3.1 gives the number of POPMUSIC solutions and the time used for reaching a state where all edges of the best known tour are included in the union of solutions.

Instance	Solutions	Time (s)	Degree
pla85900	547	203.3	14.0
mona-lisa100K	241	120.2	13.8
sra104814	322	182.7	13.0
star109399	105	64.8	15.0
usa115475	241	120.2	13.8

Table 6.3.1 Results for termination when all edges of a best known tour are included

Table 6.3.2 reports the overall performance for these instances.

	pla85900	mona-lisa100K	sra104814	star109399	usa115475
POPMUSIC Time (s)	27.5	25.8	32.1	53.8	36.5
Reduction Time (s)	18.3	21.5	24.7	26.8	28.9
Lin-Kernighan Time (s)	9483.9	3828.0	3628.6	2121.1	3858.8
POPMUSIC Degree	7.3	6.1	8.5	11.6	7.1
Reduction Degree	4.9	5.0	5.0	5.0	4.9
POPMUSIC Missing	26	0	94	10	10
Reduction Missing	61	2	388	77	110
Lin-Kernighan Gap (%)	0.056	0.015	0.068	0.023	0.031

Table 6.3.2 Overall performance for five benchmark instances
INITIAL_PERIOD = 100, MAX_TRIALS = 1000

Finally, the performance on the world TSP instance with 1,904,711 locations of the world is evaluated. The experimental results are shown in Tables 6.3.3 and 6.3.4. It is remarkable that all 1,904,711 edges of the best known tour (found by LKH) are contained in the union of only 467 POPMUSIC tours. Just one edge was missing in the union of the first 245 POPMUSIC tours.

Instance	Solutions	Time (s)	Degree
world	467	17815.4	12.7

Table 6.3.3 Results for termination when all edges of a best known tour are included

	world
POPMUSIC Time (s)	1830.9
Reduction Time (s)	550.6
Lin-Kernighan Time (s)	10558.1
POPMUSIC Degree	7.1
Reduction Degree	4.9
POPMUSIC Missing	287
Reduction Missing	2074
Lin-Kernighan Gap (%)	0.109

Table 6.3.4 Overall performance for the world instance
INITIAL_PERIOD = 100, MAX_TRIALS = 100

Note that the present evaluation has mainly used LKH's default parameter settings. Better tour quality may be achieved by other settings, such as settings that cause high-order basic moves to be used (see [2]).

6.4 Comparison of POPMUSIC with ALPHA and DELAUNAY candidate set generation

POPMUSIC has been compared with two other candidate set generation algorithms of LKH, ALPHA and DELAUNAY. The ALPHA algorithm is based on computation of minimum spanning 1-trees and is, like POPMUSIC, generally applicable. The DELAUNAY algorithm is based on Delaunay triangulation and applicable for Euclidean 2D-instances.

Table 6.4.1 gives for 9 large-scale instances the preprocessing time and number of edges missing from the best known tours. As shown, the time usage of ALPHA on these large instances is much higher than the two other algorithms. DELAUNAY is the fastest of the three algorithms, but the quality of the produced candidate sets (when measured by the number of edges missing from the best known tours) is always best for POPMUSIC.

Instance	Performance	POPMUSIC	ALPHA	DELAUNAY
E100k.0	Preprocessing Time (s)	55.7	3524.7	36.8
	Missing	**102**	192	198
C100k.0	Preprocessing Time (s)	61.4	3584.6	35.0
	Preprocessing Missing	**167**	404	401
T100k.0	Preprocessing Time (s)	53.9	3647.1	42.3
	Missing	**112**	221	384[1]
pla85900	Preprocessing Time (s)	45.8	1483.5	23.6
	Missing	**61**	92	88
mona-lisa100K	Preprocessing Time (s)	47.3	3858.1	37.4
	Missing	**2**	3	3
sra104815	Preprocessing Time (s)	56.8	3037.5	33.7
	Missing	**388**	**388**	394
usa115475	Preprocessing Time (s)	80.6	4621.6	39.3
	Missing	**77**	85	211
star109399	Preprocessing Time (s)	65.4	4294.5	-[2]
	Missing	**110**	200	-
world	Preprocessing Time (s)	2381.5	-[3]	823.4
	Missing	**2074**	-	4725

Table 6.4.1 Comparison of POPMUSIC, ALPHA and DELAUNAY
INITIAL_PERIOD = 100, MAX_CANDIDATES = 5

[1] Delaunay triangulation does not work correctly for toroidal instances
[2] Delaunay triangulation for 3D instances is not implemented in LKH
[3] The computation is too time-consuming (an estimate is 13 days)

7. Conclusions

This report has described the implementation in LKH of POPMUSIC for generating candidate sets. The implemented method can be applied to any TSP instance for which the distance between two cities can be computed in constant time. It does not make any assumptions about the problem structure. Experimental evaluation has shown that sparse high-quality candidate graphs can be produced fast. The empirical time complexity is almost linear, which makes it applicable for very large instances with millions of cities.

References

[1] Helsgaun, K.:
An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic.
Eur. J. Oper. Res., 126(1):106-130 (2000)

[2] Helsgaun, K.:
General k-opt submoves for the Lin-Kernighan TSP heuristic.
Math. Prog. Comput., 1(2-3):119-163 (2009)

[3] Taillard, E. D., Helsgaun, K.:
POPMUSIC for the travelling salesman problem.
Eur. J. Oper. Res., In Press (2018)