POPMUSIC FOR THE TRAVELLING SALESMAN PROBLEM

ÉRIC D. TAILLARD AND KELD HELSGAUN

Abstract. 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. A key point for treating large Travelling Salesman Problem (TSP) instances is to consider only a subset of edges connecting the cities. The main goal of this article is to present how to build a list of good candidate edges with a complexity lower than quadratic in the context of TSP instances given by a general function. The candidate edges are found with a technique exploiting tour merging and the POPMUSIC metaheuristic. When these candidate edges are provided to a good local search engine, high quality solutions can be found quite efficiently. The method is tested on TSP instances of up to several million cities with different structures (Euclidean uniform, clustered, 2D to 5D, grids, toroidal distances). Numerical results show that solutions of excellent quality can be obtained with an empirical complexity lower than quadratic without exploiting the geometrical properties of the instances.

1. Introduction

The travelling salesman problem [30, 24, 10] (TSP) is certainly the most studied NP-hard combinatorial optimization problem. Today, we are able to exactly solve instances with up to several thousand cities. Heuristic methods are able to find solutions for instances with millions of cities at a fraction of the percent above the optimum with a reasonable computational effort (see e.g. [3, 5, 19, 14, 25]).

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 few connections for each city. In the case of geometrical problems, several techniques have been proposed for generating an adequate network. For 2-dimensional Euclidean instances, a Delaunay triangulation [11] can be built in $O(n\ log(n))$, 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 KD-tree [6] (in $O(Kn\ log(n))$) and to keep only few of the nearest cities in every quadrant. This ensures to build a connected and sparse network.

The Quick-Boruvka heuristic is commonly used to build a tour. It produces moderately good solutions (generally more than 10% above optimum) very rapidly (around one second for a 1 million cities TSP). For getting better solutions, almost all the state-of-the-art methods make use of local searches based on variants of ejection chains. The oldest and most famous ejection chain method was proposed in 1973 by Lin and Kernighan [18]. The Lin-Kerhighan neighbourhood (LK for short) can be improved by enriching it with k-opt moves, consisting of replacing k edges by

Key words and phrases. Travelling Salesman, Local search, POPMUSIC, Large-Scale Optimization, Metaheuristics.

June 22, 2018 ©2018 This manuscript version is made available under the CC-BY-NC-ND 4.0 license. DOI: 10.1016/j.ejor.2018.06.039.

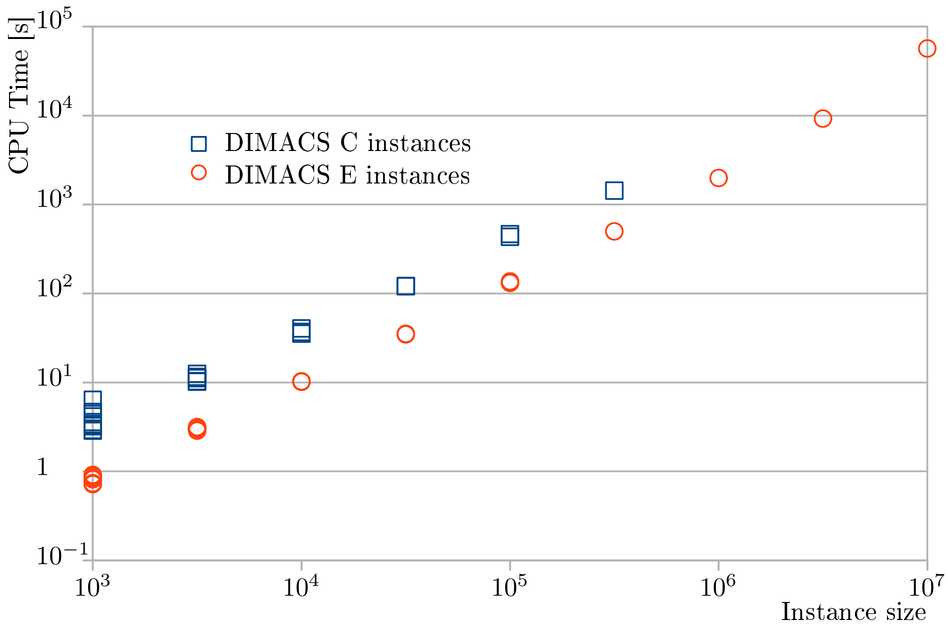

FIGURE 1. Computational time of *Concorde linkern* on uniform (E) and clustered (C) DIMACS instances with default parameters.

k other ones. For instance, the LKH code available on the web site [14, 15] (`http://webhotel4.ruc.dk/~keld/research/LKH/`) incorporates 5-opt moves in addition to the LK neighbourhood. Better solutions can be obtained by perturbation of the local optimum. This is done for instance in the *Chained Lin-Kernighan* of [5] implemented in *Concorde* [3] (`http://www.math.uwaterloo.ca/tsp/concorde.html`) by applying double-bridges 4-opt moves. A perturbation mechanism is also implemented in the LKH code, but with different moves. In this code, each LK restart is called a trial.

Figure 1 reports the computational time and as a function of the instance size for the *linkern* code of *Concorde*. The Euclidean instances considered in this figure are those of the DIMACS TSP Challenge [16] uniformly distributed in a 10000×10000 square (E type) and clustered instances (C type). Throughout this article, the computational times are expressed in seconds for a computer running Linux 4.4.0-31-generic(x86-64), using a single core of an i7 930 processor @2.8GHz (commercialized in March 2010). We see in Figure 1 that the computational time increases almost linearly with the instance size. Regressions on computational times provide $t(n) \approx 1.64 \cdot 10^{-4} \cdot n^{1.05}$ for C instances and $t(n) \approx 1.66 \cdot 10^{-5} \cdot n^{1.21}$ for E instances. Experiments on TSPLIB instances have shown a very similar behaviour.

Figure 2 reports the quality of the solutions obtained for *linkern* and LKH. In this figure, the deviation is expressed in percent above the best values that are known, published on the web site `http://webhotel4.ruc.dk/~keld/research/LKH/DIMACS_results.html`. For instances with less than 10,000 cities, the proven optima are known. LKH was run with slightly different parameters than default ones: CANDIDATE_SET_TYPE = DELAUNAY PURE, EXTRA_CANDIDATES = 4, INITIAL_TOUR_ALGORITHM = QUICK-BORUVKA, SUBGRADIENT = NO. Figure 2 reports the solution produced by LKH after the same computationnal effort as *Concorde linkern*. Both methods were run only once for each instance for showing the spread in computational time and solution quality for the

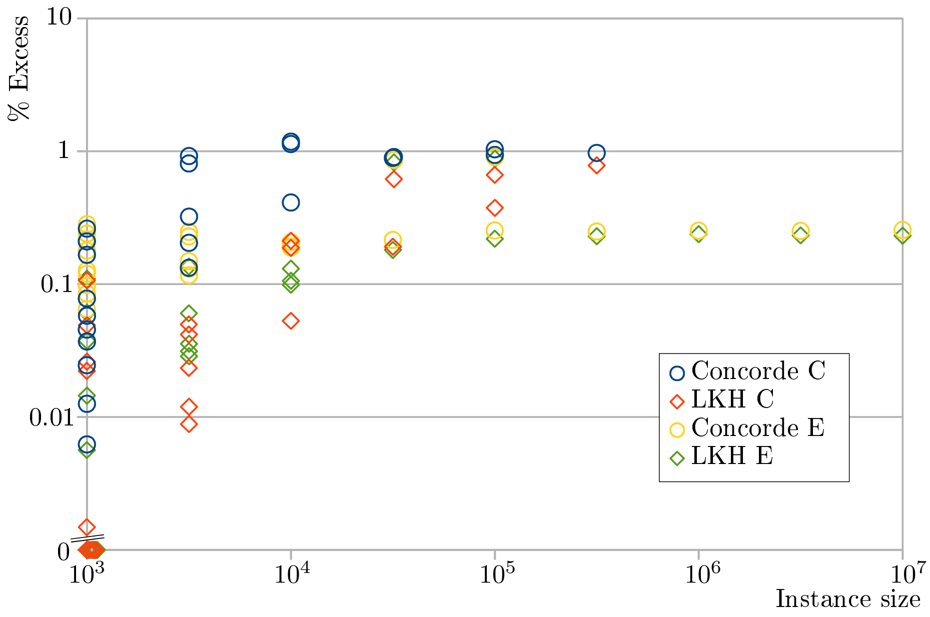

FIGURE 2. Quality of solutions produced by *Concorde linkern* and LKH on uniform (E) and clustered (C) DIMACS instances. LKH is run with default parameters but CANDIDATE_SET_TYPE = DELAUNAY PURE, EXTRA_CANDIDATES = 4, INITIAL_TOUR_ALGORITHM = QUICK-BORUVKA, SUBGRADIENT = NO. LKH is stopped at the same CPU times as those observed for *Concorde linkern*

same instance size. We see in this figure that LKH generally provides slightly better solutions than *Concorde linkern*. We have observed that LKH has produced a slightly worse solution for only one of the 49 DIMACS C and E instances. All leading methods for the TSP are able to produce similar results. The reader can see [25] for a comparison of leading methods for the TSP.

Several practical applications can lead to TSPs for which geometrical properties cannot be directly exploited. Among them, let us quote no-wait flowshop scheduling, one machine scheduling with set-up time, DNA sequencing, X-ray crystallography, colour 3D printing with minimization of colour changes, drone surveying taking into account elevation, wind strength and energy, and, generally, asymmetrical TSP instances. See, e.g. [8] for such instances modelling real-world applications. The 3D printing and drone surveying are new applications that can lead to very large instances. The computational effort for producing good solutions for such instances must be kept very limited.

Concorde and LKH can deal with such instances, but the matrix of all distances between each pair of cities must be provided. This limits the instance size that *Concorde* can treat. Problem instances with 31k cities cannot be treated with this code with distance matrix (while this remains possible with the LKH code). For creating a sparse network for applying a local search, *Concorde* considers a fixed number of the nearest cities for each city. The draw-back of this method is that the resulting sparse network may not be connected. This may degrade the quality of the solutions produced. When providing the full distance matrix, for the DIMACS C instances with $\leq 10,000$ cities, *Concorde linkern* produces solutions that are, on

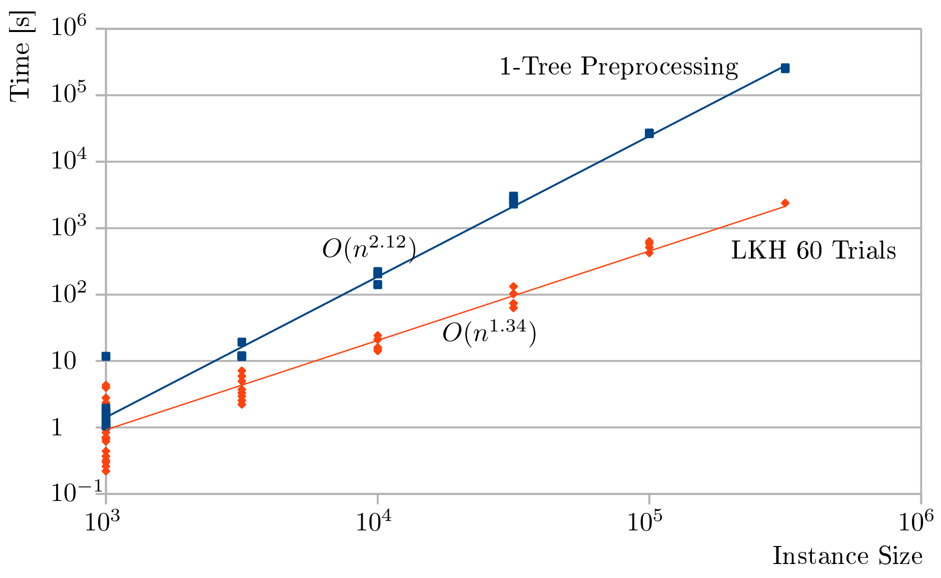

FIGURE 3. Computational time of LKH for uniform (E) and clustered (C) DIMACS instances, with candidate edge list found with LKH's standard preprocessing, 60 trials and a single run. The lines correspond to interpolation polynomials.

the average, 2.6% above those obtained when providing the same instances with Euclidean coordinates.

If not specified otherwise, LKH creates a sparse network by computing a 1-tree (a minimum weight spanning tree on all vertices but one plus 2 edges adjacent to this vertex) and uses a concept named α-nearness (minimum modification of the penalty associated to a vertex in order to change its degree in the 1-tree). The resulting sparse network is connected, but not necessarily Hamiltonian (see e.g. Figure 15).

This method can be applied to any symmetrical TSP but requires a computational time that increases quadratically, as shown on Figure 3.

When the distance between 2 cities is given by a general function (that can be computed in constant time), it is challenging to find a good tour with a computational effort that increases less than quadratically. Obtaining a solution with the nearest neighbour greedy heuristic, which is faster than Quick-Boruvka's one, would take several days of computation for a 10 millions cities instance. Indeed, these heuristics, or a pre-treatment procedure, require to compute all the distances, meaning an algorithmic complexity in $O(n^2)$ for a problem with n cities.

Blazinskas and Misevicius have proposed a method for generating good candidate sets by a technique called *tour merging* [9, 4, 7]. The method of [7] consists of generating several solutions with a multi-random-start local search procedure. The local search is based on 3-opt moves. It is speeded up by considering the 40 nearest neighbours for each vertex, 10 in each quadrant. If implemented for non geometric problem instance, the complexity of the tour merging techniques of [9, 4] are at least quadratic.

The main goal of this article is to present how to build a list of good candidate edges with a complexity lower than quadratic in the context of problem instances given by a general function. The candidate edges are found with a technique exploiting tour merging and the POPMUSIC metaheuristic [29]. When these candidate edges are provided along with an initial tour to a good local search engine like LKH, high quality solutions can be found quite efficiently.

The paper is organized as follows: Section 2 presents the POPMUSIC metaheuristic and shows how it can be used in the context of the TSP. Section 3 presents a large neighbourhood search (LNS [26, 23]) method for the TSP. The idea of LNS is to gradually improve an initial solution by alternately destroying and repairing the solution. Section 4 presents how to efficiently generate candidate edges, allowing the LKH implementation to rapidly produce very good solutions.

Computational results are presented in Section 5. The method is tested on various problem types for which the value of optimum solutions are known or have been estimated with a good precision: Euclidean 2D, clustered, uniform with toroidal distances in 2D, 3D and 4D and regular grids in 5D. Such instances have been used since they are the only ones of large size that are available or that can be easily generated. Although these instances are geometrical ones, the proposed method does not exploit their geometrical properties. The goal of this paper is to show that good solutions to large instances can be obtained without exploiting the geometrical properties, not to solve faster or get better solutions to such instances than existing methods of the literature exploiting the geometrical properties. Concluding remarks and future research avenues are presented in the last section.

2. POPMUSIC METAHEURISTIC

The POPMUSIC metaheuristic was formalized in [29] on the base of work going back to the early 1990s [27]. The basic idea of POPMUSIC is to locally optimise sub-parts of a solution after a solution of the problem is available. These local optimisations are repeated until no improvements are found.

For the TSP, the first idea coming to mind is to optimise the connections inside a group of cities that are the closest to a given city. This option is not good for TSPs with general distances. Indeed, identifying such a group would take a computational effort in $O(n)$. Since there are n different groups to consider, the global effort would be in $O(n^2)$. This is prohibitive for large instances.

So, we propose to optimise sub-paths of r consecutive cities. A tour on n cities can be considered as a set of $s_1, \dots, s_n$ sub-paths, each containing r cities. Sub-path s_i contains the ith, $i+1$th, $i+r$th cities (modulo n). This definition allows to easily identify a sub-part of a solution. An optimised sub-path can be easily replaced in the solution for improving it.

To avoid generating the same sub-problem two times, a set U of sub-paths is stored (just by storing the first city of each sub-path). U contains sub-paths of r cities that potentially can be improved. Once U is empty, all sub-problems have been examined without success and the process stops.

Algorithm 1 presents the POPMUSIC metaheuristic with our adaptation to the TSP.

This metaheuristic can, potentially, have very high complexity since set U is not reduced at each iteration. However, several implementations [28, 21, 1, 2] have shown empirically that the number of iterations of an algorithm based on this metaheuristic increases almost linearly with the number of parts constituting a solution. If the size r of the sub-problems to be optimised in POPMUSIC is fixed, the empirical complexity of the method is almost linear in instance size.

Therefore, a major challenge when implementing a POPMUSIC-based algorithm is building the initial solution with an algorithmic complexity as low as possible. The solution need not be of very high quality, but its structure must be convenient for being partially re-optimised.

Algorithm 1: POPMUSIC metaheuristic for the TSP

```
Data: Initial tour $T$ composed of $n$ parts $s_1, \dots, s_n$
Result: Improved solution $T$
1 $U = s_1, \dots, s_n$;
2 while $U \neq \emptyset$ do
3     Select $s_s \in U$;
4     Try to optimise the sub-path $s_s$ of $r$ consecutive cities of the tour, fixing
      the first and last cities of the sub-path;
5     if Sub-path has been improved then
6         Update solution $T$;
7         Add or replace in $U$ all the sub-paths that have been modified
8     Remove $s_s$ from $U$;
```

Intuitively, since we have chosen to optimise sub-paths of a solution, the last should avoid to contain two sub-paths that are located in completely different portions of the tour but containing cities that are close each others. Indeed, the separate optimisation of these sub-paths will not allow to re-organize correctly their cities.

To translate the POPMUSIC metaheuristic into a (pseudo-) code for a given problem, several options must be chosen:

- How to obtain the initial solution
- Which optimisation method to use for optimising sub-paths
- Which sub-path from U to select

2.1. Obtaining an acceptable initial solution.

Algorithm 2 is used for getting an initial solution that is convenient for POPMUSIC. This algorithm can be implemented with a relatively low complexity and avoid to produce solutions with very long edges. Indeed, greedy methods like nearest neighbour or Burovka may include very long edges during their last iterations.

Algorithm 2: Generating a feasible solution for large instances

```
Data: $n$ cities, distance function $d(i, j)$ between cities $i$ and $j$, parameter
      $0 < a < n$
Result: TSP tour $T$
1 Select a sample $S$ of $a$ cities, randomly, uniformly, among the $n$ cities;
2 Build a LK-optimal tour $T_s$ on $S$;
3 $T = T_s$
4 for each city $c \notin S$ do
5     $c_s = argmin(d(s, c)), (s \in S)$;
6     Insert $c$ just after $c_s$ in $T$;
7 for each city $c \in T_s$ do
8     Let $n_c$ be the number of cities inserted at previous step after $c$ and after
      the city next to $c$ in $T_s$;
9     Optimise in $T$ a sub-path of $n_c$ cities starting from $c$ with a 2-opt local
      search
```

The main steps of this algorithm are illustrated in Figure 4. In this figure, the tour on the sample S is given in bold and light colour. The complete tour is given by a narrow, darker line. The complete tour is partially optimised, in a situation where the last loop of Algorithm 2 is partially performed.

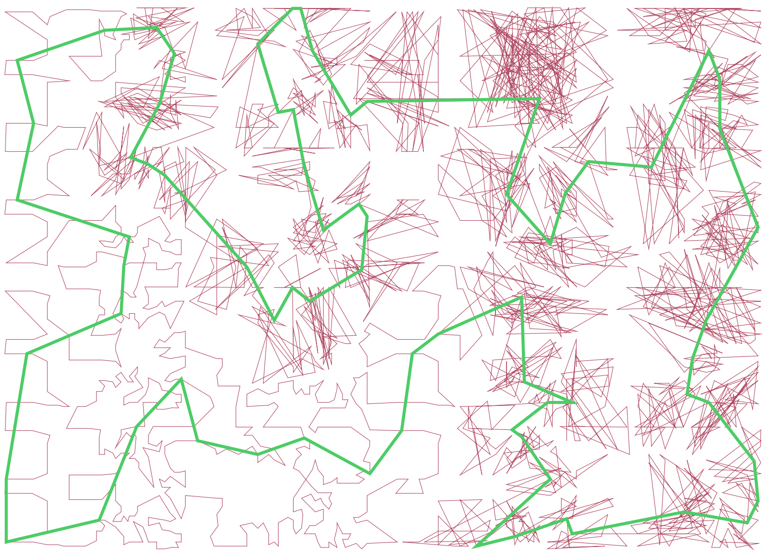

FIGURE 4. Building an initial solution adapted for `POPMUSIC`. First, a tour is found on a sample of cities (bold line). Then, the remaining cities are inserted after the closest of the sample, creating clusters of cities. Finally, sub-paths including the cities assigned to 2 consecutive clusters are optimised. The figure shows the situation when the cluster optimisation is partially performed.

2.1.1. *Optimising sub-paths.* Line 9 of Algorithm 2 as well as Line 4 of Algorithm 1 require procedures for optimising sub-paths. Sub-path optimisation can be transformed into a TSP as follows: let r be the number of cities of a sub-path of length L and d_{ij} the distance between cities i and j. Finding the shortest way to connect the first and last cities of the path while visiting once all the cities of the path can be solved as a TSP defined with distance matrix $\mathbf{D}$, where index 1 and r refer to the first and last city in the subpath:

$$\mathbf{D} = \begin{pmatrix} 0 & d_{12}+L & d_{13}+L & \dots & 0 \\ d_{21}+L & 0 & d_{23} & \dots & d_{2r}+L \\ d_{31}+L & d_{32} & 0 & \dots & d_{3r}+L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & d_{r2}+L & d_{r3}+L & \dots & 0 \end{pmatrix}$$

The idea is to fix an edge between 1 and r by setting its length to 0. The other edges from and to 1 and r are penalized by a relatively large value L.

2.1.2. *Building a LK-optimal tour on a sample of the cities.* For finding a LK-optimal solution on the sample at Line 2 of Algorithm 2, the LKH code could be an alternative. However, this code is not very convenient to use inside another program because it has been designed to be stand-alone and not callable. Preliminary results have shown that it is not very useful to use leading methods either for solving TSP on a sample of the cities or for optimising sub-paths.

To avoid implementation issues, we have used a very basic local search, based on the LK neighbourhood, without building a graph of candidate edges (all edges are retained). This algorithm uses a data structure called "satellite-cities" by [20]. This data structure can reverse a sub-path in constant time. This procedure can be written in about a hundred lines of C code. Its empirical algorithmic complexity is

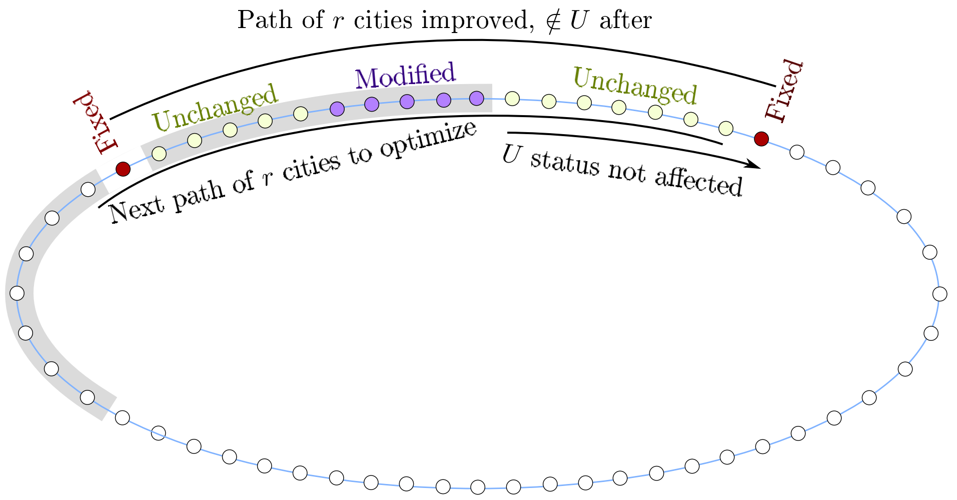
Path of r cities improved, $\notin U$ after
Fixed
Unchanged
Modified
Unchanged
Fixed
Next path of r cities to optimize
U status not affected

FIGURE 5. After having optimised a sub-path of r cities, it can be removed from U, since it is already LK-optimal. The 2 grey areas identify the starting cities of sub-paths that must be included in U. If the sub-path is modified, U can be managed in such a way that the next sub-path to optimise is shifted only by one city.

slightly less than cubic ($O(n^{2.78})$). By choosing a relatively small sample size ($a \sim n^{0.56}$, see Section 2.3), this procedure remains faster than the LKH implementation. This procedure is also used for optimising sub-paths at Line 4 of Algorithm 1.

2.1.3. *Optimising sub-paths of 2 clusters.* The loop at Line 4 of Algorithm 2 creates clusters of cities that are connected by sub-paths arbitrarily constructed. Line 9 requires optimising a times a relatively long sub-path that contains, on average, $2n/a$ cities. Using the above procedure would be too time consuming. Therefore, we chose to optimise the paths that corresponds to 2 clusters of consecutive cities with a simple local search based on a 2-opt neighbourhood. This local search also uses the satellite cities data structure and has an empirical algorithmic complexity in $O(n^{2.29})$.

2.2. **Selection of the next sub-path to optimise in `POPMUSIC`.** In preceding `POPMUSIC` implementations for the VRP, map labelling or location-routing problems, the next sub-problem to optimise was randomly chosen. For the TSP, the process can be slightly speeded up by managing set U in such a way that the next sub-path to optimise is shifted by only one city in case a modification occurs. Indeed, by treating first the last sub-path of cities optimised, one gets a longer and longer portion of the complete tour for which all sub-paths of r consecutive cities are LK-optimal.

When a sub-path containing r cities has been successfully optimised, the sub-paths with starting cities that remain in the same order after optimisation need not to be added in U. Indeed, these cities will belong to another sup-path to optimise in the future. Figure 5 illustrates the management of set U.

2.3. **Minimising `POPMUSIC` algorithmic complexity.** For the TSP, the number of sub-problems to optimise in the `POPMUSIC` metaheuristic increases empirically almost linearly with instance size (see Figure 7). So, getting an initial solution with Algorithm 2 contributes most to the algorithmic complexity. It can be derived as follows: let us first suppose that the complexity of finding a tour on a sample

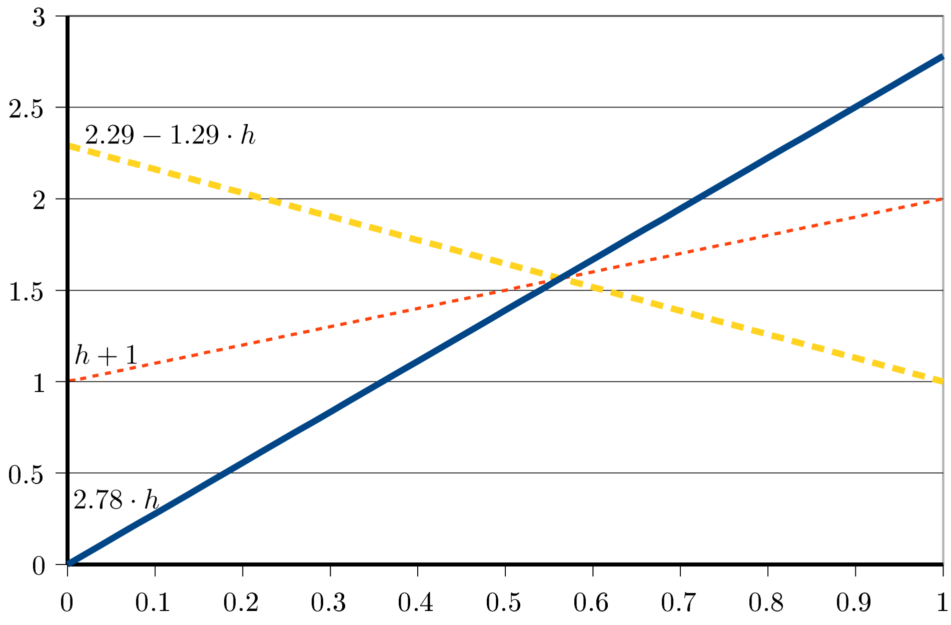

FIGURE 6. Degree of the polynomial for various steps of initial solution generation as a function of h. The sample size is supposed to be in $O(n^h)$, finding a 2-opt solution in $O(n^{2.29})$ and a LK local optimum solution in $O(n^{2.78})$.

of a cities is in $O(a^f)$ and second that the complexity for optimising sub-paths of $2n/a$ cities (on the average) is in $O((\frac{n}{a})^g)$. Then, the algorithmic complexity of Algorithm 2 is in $O(a^f + n \cdot a + a \cdot (\frac{n}{a})^g)$. The term $n \cdot a$ comes from the fact that each of the $n - a$ cities not in the sample must identify the city of the sample that is the closest.

Therefore, one has to find a sample size a, expressed as a function of n, that minimises the complexity of Algorithm 2. Choosing $a = n^h$, the complexity of the algorithm can be written as $O(n^{fh} + n^{h+1} + n^h \cdot n^{g-hg})$. Minimising this complexity implies minimizing $max(fh, h + 1, g + (1 - g)h)$.

Figure 6 provides the values of the 3 terms of this maximum as a function of h, taking into account the empirical complexity of our LK-opt and 2-opt implementations, which are $f \simeq 2.78$ and $g \simeq 2.29$. It can be seen that the value of h minimising the maximum complexity is about 0.56. For $h = 0.56$, the value of the maximum is 1.57. So, it is possible to implement Algorithm 2 with an empirical complexity proportional to $n^{1.57}$.

Using local search optimisation procedures with a lower complexity (for instance, LKH with parameters such that $f \simeq g \simeq 2$) the maximum can be lowered up to 1.5 for $h = 0.5$. However, we have observed that, for problem instances whose size is lower than few dozens of millions, the time spent in running LKH is higher than the time spent in our procedures. Conversely, it is possible to use procedures for optimising sub-paths with a higher complexity. While this complexity is such that $f, g < 2 + \sqrt{2}$, producing an initial solution has a complexity lower than $O(n^2)$.

2.4. **Empirical complexity of the algorithms proposed.** For evaluating the empirical complexity of the algorithms proposed, we have chosen samples of size $a = 1.5 \cdot n^{0.56}$ for generating initial solutions. For the `POPMUSIC` algorithm, we have chosen to optimise sub-paths of $r = 50$ cities. The algorithms were run on

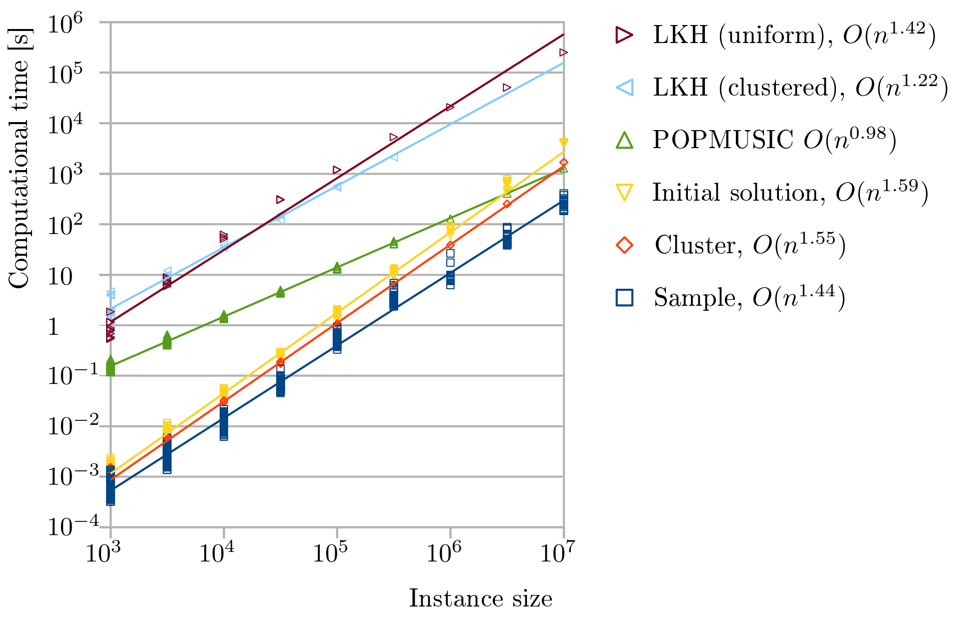

FIGURE 7. Computational time for DIMACS instances as a function of instance size for each step of the proposed method. Tour on a sample (Line 2 of Algorithm 2), creating clusters (Line 4), generating the initial solution (Line 7) and optimising this solution with POPMUSIC (Algorithm 1). The lines are interpolated polynomials, whose degree is indicated in the caption. The correlation coefficients of the interpolations are over 0.99 in all cases.

DIMACS problem instances of type E (cities uniformly distributed on a square) and C (clusters of cities on the Euclidean plane). Figure 7 provides the computational time as a function of the instance size, for every step of the algorithm. Every problem instance was solved 20 times.

We see in this figure that building a tour on a sample of cities has the most versatile computational time (and is also the fastest step) and the empirical complexity of generating an initial solution ($n^{1.61}$) is very close to the complexity predicted in the previous section ($n^{1.57}$). Re-optimising a solution with `POPMUSIC` seems to be linear. For the smallest instances, `POPMUSIC` is 100 times slower than producing an initial solution, while it is 10 times faster for the largest instances.

3. Large neighbourhood search for the TSP

Compared to state-of-the art algorithms such as LKH (see Figure 3), the algorithm proposed in this article is quite fast for non-Euclidean instances. Naturally, this speed must be considered in tandem with the quality of the solutions. It is clear that methods limited to Euclidean instances can be much faster (for instance those working on a space-filling curve [22] or a Delaunay triangulation). Although our method can be applied to any problem instance, it will certainly work very poorly on instances with random distances.

3.1. **Tour recomposition.** A situation that may occur with our method is that 2 cities, relatively far from each other, are connected because they belong to two successive clusters that are separated by a third cluster visited in a completely different portion of the tour. This undesirable situation is illustrated in Figure 8.

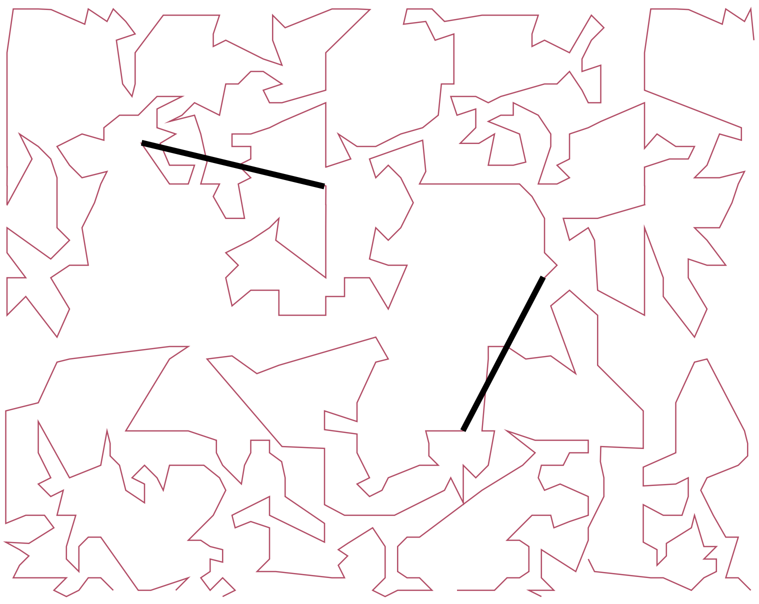

FIGURE 8. Example of occurrence of 2 bad and long edges in a portion of a solution produced by POPMUSIC .

This could be prevented by trying to re-optimise sub-paths containing more cities at line 9 of Algorithm 2 for getting the initial solution. For instance, one could try to optimise sub-paths containing the cities of 6 or 7 clusters rather than 2. In such a case, the computational time is multiplied by a factor higher than 10.

Another approach is to detect the use of long edges and to suppress them. The TSP tour is separated out in a set of sub-paths. A possibly better solution can be built by connecting these sub-paths differently. Sub-paths can be reconnected together into another TSP tour as follows:

- Sub-paths containing 3 cities or less are replaced by isolated cities
- Sub-paths containing more than 3 cities are replaced by 3 cities

The distances between the cities of the new TSP are the same as the original one in the first case. For the second case, let us call b the first city of a sub-path with more than 3 cities and e the last city of this sub-path. Let us call r, s and t the 3 cities that will replace the sub-path and i any city in the original problem. Finally, let M be a large value and d_{be} (d_{eb}) the length of the sub-path from b to e (respectively: from e to b). Then, the distances between cities i, r, s and t are given by the next table.

	i	r	s	t
i	—	$d(i,b)$	M	$d(e,i)$
r	$d(b,i)$	—	d_{be}	M
s	M	d_{eb}	—	0
t	$d(e,i)$	M	0	—

This tour recomposition can be seen as a large neighbourhood search [26, 23] (this destroy-and-repair method is also called *fix-and-optimise* [12]). For implementing this method, the edges to remove must be chosen. An option is to suppress the longest edge of each sub-path of p consecutive cities in the initial TSP tour. For reconnecting the sub-path into a complete TSP tour, a LK-based local search can be used. When choosing $p = n^{0.44}$, we have observed that the size of the new

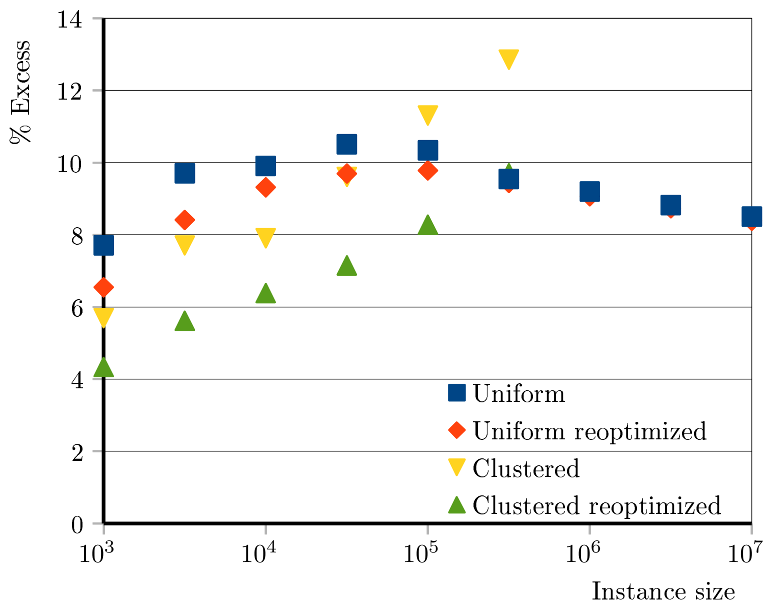

Figure 9. Quality of solutions obtained with `POPMUSIC` for DIMACS uniform and clustered instances either without or with re-optimisation of the solution.

TSP increases as $O(n^{0.55})$ on DIMACS C and E instances. This means that the re-optimisation of a TSP with this method takes no more time than the other steps of generating a complete TSP tour.

After having re-optimised a TSP tour with this large neighbourhood search, `POPMUSIC` can be reapplied to the new solution while improvements are found. In practice, we have observed that this cycle is repeated only a few times in practice (less than half a dozen).

Figure 9 gives the average deviation obtained by `POPMUSIC` with or without re-optimisation, for DIMACS E and C problem instances. We see that for clustered instances, an improvement of 2 to 3% can be obtained. For uniform instances, this improvement is lower and asymptotically tends to 0. If better solutions of good quality must be obtained rapidly, another technique must be used.

4. Generating candidate edges with tour merging

A well known technique to drastically reduce the search space for the TSP is to consider a very small number of possible connections for each city. This is true both for exact or heuristic methods. As a solution produced by `POPMUSIC` issues from a highly randomised process, several successive runs of `POPMUSIC` generate several different solutions. Repeating this a few dozen times gets a subset of pertinent connections for building high quality tours.

Figure 10 presents a moderately good solution obtained by `POPMUSIC` (without using the improvement with a large neighbourhood search presented at the previous section) as well as the union of 20 different solutions obtained by this method for the TSPLIB problem instance pr2392 (with 2392 cities). One optimal solution is superimposed in the figures. We see in Figure 10 that a `POPMUSIC` solution contains a lot of edges that belong to an optimal tour. The union of 20 different solutions contains all but a couple of a given optimal tour edges.

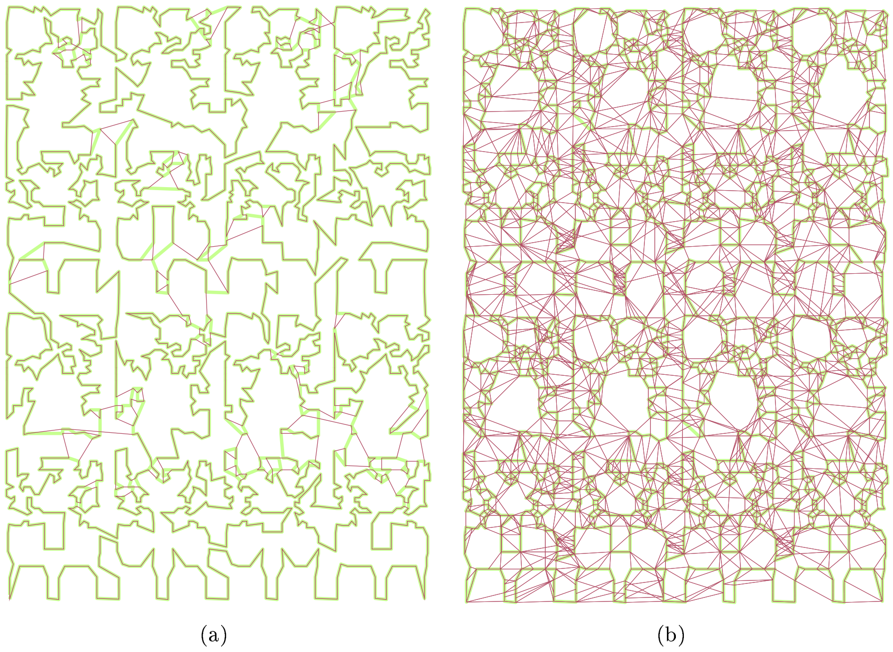

FIGURE 10. Solution obtained with a POPMUSIC run 10(a). Union of 20 solutions found by POPMUSIC 10(b). One optimal solution is indicated by bold and light colour lines. A solution obtained with POPMUSIC, although being not very good, contains a high proportion of the edges of the optimal tour. The union of 20 solutions contains almost all the edges of the optimal tour.

Figure 11 provides an upper bound to the statistical distribution of the number of missing edges as a function of the number of POPMUSIC solutions generated for TSPLIB instance pr2392. Let us mention here that this instance has several optimal tours. Two different optimal tours may differ by 60 edges [1]. In Figure 11, the number of missing edges is counted relatively to the optimum solution 1 . . . 2392.

For generating this figure, hundreds of solutions were generated with POPMUSIC. Then, the statistical distribution for various number of solutions generated is estimated with a bootstrap technique. In this figure, the distribution is indicated by modulating the colour density. A dark colour indicates a high probability. This figure suggests that this technique can rapidly produce a subset of edges that are good candidates for building excellent quality TSP tours.

The average degree of the vertices in the graph of Figure 10(b) is 6.58. This degree is to compare to the average degree of a Delaunay triangulation which is slightly lower than 6. For this problem instance, the optimal TSP tour has more edges that do not belong to the Delaunay triangulation than edges that do not belong to the union of 20 POPMUSIC solutions. Figure 12 provides the statistical

[1]One optimal tour is 1 . . . 2392. Another one is 1 . . . 246 249 . . . 253 248 247 254 . . . 315 317 316 318 . . . 364 366 365 367 . . . 421 424 . . . 465 472 471 466 . . . 470 473 . . . 605 607 606 608 . . . 655 657 656 658 . . . 704 706 705 707 . . . 768 775 774 769 . . . 773 776 . . . 960 422 423 961 . . . 1030 1032 1031 1033 . . . 1165 1172 1171 1166 . . . 1170 1173 . . . 1207 1214 1215 1208 . . . 1213 1216 . . . 1400 1407 1406 1401 . . . 1405 1408 . . . 1469 1471 1470 1472 . . . 1518 1520 1519 1521 . . . 1675 1678 . . . 1683 1676 1677 1684 . . . 1718 1725 1724 1719 . . . 1723 1726 . . . 1858 1860 1859 1861 . . . 1908 1910 1909 1911 . . . 1957 1959 1958 1960 . . . 2021 2028 2027 2022 . . . 2026 2029 . . . 2148 2150 2149 2151 . . . 2283 2290 2289 2284 . . . 2288 2291 . . . 2325 2332 2333 2326 . . . 2331 2334 . . . 2392

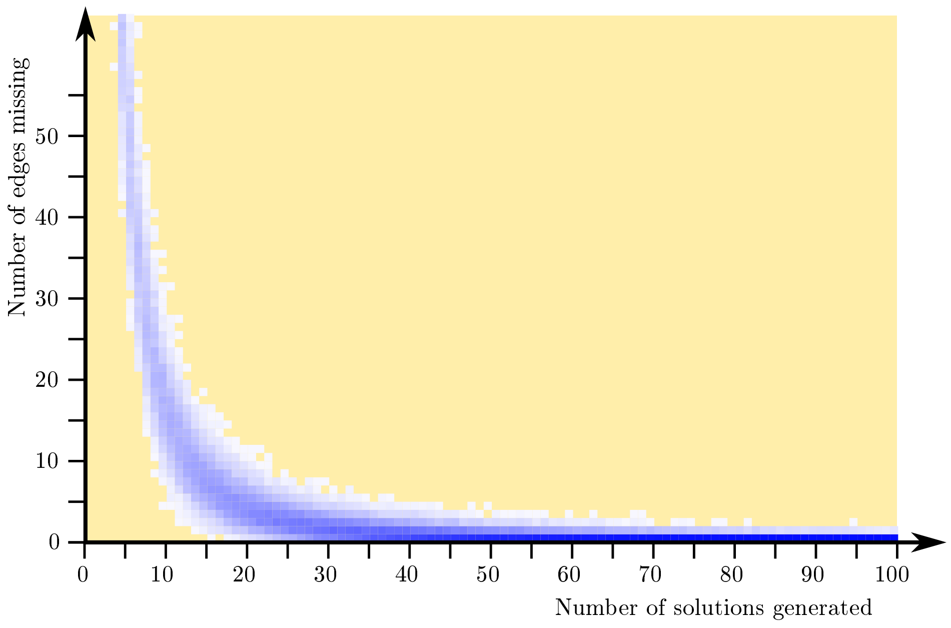

FIGURE 11. Distribution of the number of edges of the optimal solution not in the union of solutions generated with POPMUSIC, as a function of the number of solutions generated. A dark colour indicates a high probability distribution.

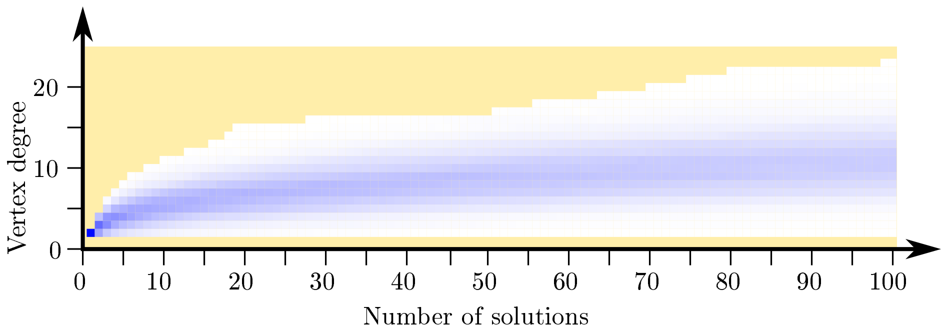

FIGURE 12. Evolution of vertex degree as a function of the number of solutions generated with POPMUSIC. A darker colour indicates a higher number of vertices.

distribution of the vertex degree of the union of POPMUSIC solutions as a function of the number of TSP solutions generated, for TSPLIB problem instance pr2392. We see in this figure that the average degree increases very slightly with the number of solutions generated with POPMUSIC.

Figure 13 provides the distribution of the vertex degree for various problem instances from the DIMACS library. We see in this figure that the vertex degree depends more on the instance type (clustered, uniform) than on the instance size (from 10^5 to 10^7 cities).

In order to more deeply analyse the characteristics of the sub-graph induced by the union of 20 solutions produced by POPMUSIC, problem instances in larger dimensions have been considered: instances randomly generated on the unit hypercube in

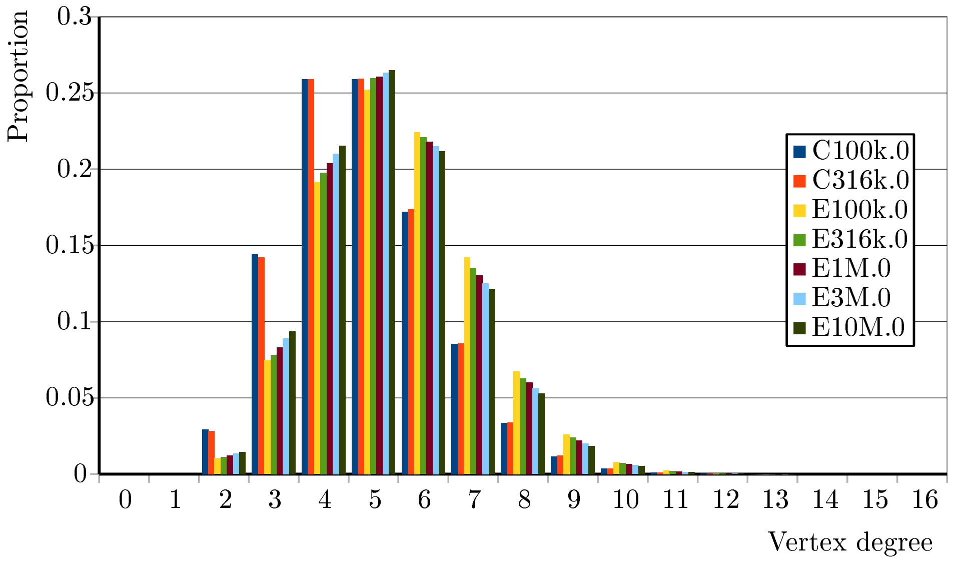

FIGURE 13. Degree of vertices for few DIMACS problem instances for graphs that are the union of 20 solutions generated with POPMUSIC.

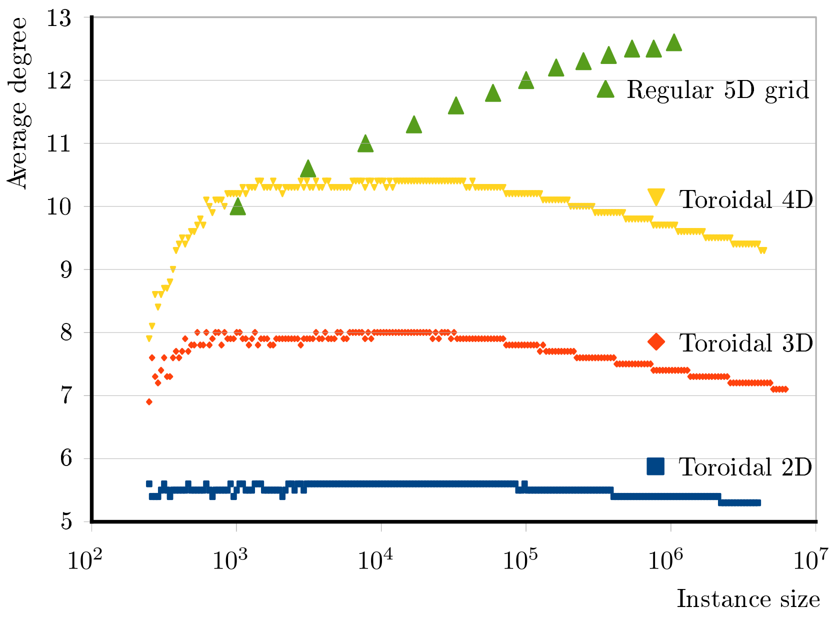

FIGURE 14. Average degree of vertices for graphs that are the union of 20 solutions generated with POPMUSIC. Uniformly distributed instances with toroidal distances in 2D, 3D and 4D and regular grid instances in 5D.

dimensions 2, 3, and 4 with toroidal distances (the opposite sides of the hypercube are identified) and instances on a regular, complete grid with integer coordinates in an hypercube of dimension 5. For complete grids, the optimal solution length is just equal to the instance size.

Figure 14 provides the average degree of the vertices of the sub-graph induced by the union of 20 solutions produced by `POPMUSIC` as a function of instance size. We see in this figure that the average degree is almost independent of the instance size for randomly generated instances of size larger than 1000. However, the degree depends strongly on the instance dimension.

This can be intuitively explained considering instances on regular grids in dimension K. For such instances, all vertices (except those on the border) have $2K$ neighbours at distance 1. Thus, a pertinent sub-graph should have an average vertex degree of $2K$. For large instances with $K = 5$, we see in Figure 14 that the degree is approximately 12.5. This means that `POPMUSIC` generates inaccurate edges about 25% of the time for these instances. However, it is not clear why the average degree decreases for toroidal instances of size higher than 10^5.

5. Numerical results

In the previous sections, we have seen that it is possible to generate sub-graphs with a low algorithmic complexity. This section studies the pertinence of the edges generated.

To evaluate this, the LKH code has been adapted with the ability to directly read a set of edges with their weight (including an EDGE_FILE parameter). For all the numerical results presented in this section, we have chosen sample sizes of $1.5{\cdot}n^{0.56}$, sub-paths of 50 cities in `POPMUSIC`. The recomposition technique presented in Section 3 has not been used. LKH is executed considering the edges obtained by the union of 20 `POPMUSIC` solutions and with the following parameters:

- SUBGRADIENT = NO
- MAX_CANDIDATES = 0
- EDGE_FILE = ...
- INITIAL_TOUR_FILE = ...
- MAX_TRIALS = 60
- RUN = 1

The initial tour given to LKH is one of those obtained with `POPMUSIC`. Experiments not reported here have shown that there is no clear correlation between the quality of the initial solution provided and the final solution produced by LKH. Depending on the instance, the correlation might even be slightly negative.

Tables 1 and 2 compare the solution quality and the computational time of the proposed method versus the original version of LKH working with a sub-graph obtained with 1-trees (standard parameters but: MAX_TRIALS = 60 and RUN = 1). It must be stressed here that LKH produces much better results if used with parameters exploiting the Euclidean property of the instances (see Figures 1 and 2). The 1st column gives the TSP size, the 2nd the optimum solution value (for instances up to 3162 cities) or best solution known (published on [13]), the 3rd is the computational time for producing 20 solutions with `POPMUSIC`, the 4th is the total computational time of the proposed method, the 5th is the computational time of standard LKH running with 1-trees and the last 2 columns compare the solution quality obtained by the proposed method and LKH, expressed in % over best known.

As shown in Figure 7, most of the computational time for standard LKH is spent for finding candidate edges with 1-trees. As the computational time was prohibitive for instances with more than $10^{5.5}$ cities, we have not run LKH with standard parameters for these instances. For C instances, the preprocessing phase of LKH was speeded up by setting INITIAL_PERIOD = 1000.

For uniform instances (Table 1), we see that running LKH using sub-graphs with regular vertex degree of 5 obtained with 1-trees seems to be slightly better than

Instance size	Best known	Computational Time [s]			%Excess	
		Sub-graph	Total	LKH	Proposed	LKH
1000	23101545.4	2.981	3.921	1.701	0.036	0.016
3162	40519926	9.406	17.54	16.4	0.062	0.022
10000	71865826	29.7	92.2	226	0.11	0.05
10000	72031630	29.9	82.3	225	0.08	0.03
10000	71822483	29.6	87.0	237	0.12	0.03
31623	127282138	95.3	427	2396	0.19	0.06
31623	126647285	95.2	470	2427	0.83	0.73
100000	224330692	325	1850	26860	0.88	0.76
100000	225654639	326	1805	27446	0.24	0.11
316228	401301206	1234	7821	262090	0.26	0.15
1000000	713187688	5250	31061	—	0.27	—
3162278	1267318198	27770	129866	—	0.28	—
10000000	2253088000	208195	611402	—	0.28	—

Table 1. Results for uniform (E) DIMACS instances. Computational time for producing 20 solutions with POPMUSIC, total time of the method (LKH with 60 trials working of union of 20 POPMUSIC solutions), time of LKH with standard parameters (not exploiting the Euclidean property of the instances, 1 Run with 60 trials). Quality of solutions produced by our method (% excess over best solution known) and by standard LKH. For problems of size 1000 and 3162, the results are averaged for 10 (respectively: 5) instances.

Instance size	Best known	Computational Time [s]			%Excess	
		Sub-graph	Total	LKH	Proposed	LKH
1000	11174460.5	3.24	4.18	4.13	0.11	0.10
3162	19147233.6	10.5	20.3	15.1	0.68	1.50
10000	33001034	33.1	85.4	68	0.72	3.31
10000	33186248	33.4	76.8	73.6	0.86	1.42
10000	33155424	33.2	71.1	73.9	0.25	0.42
31623	59545390	108	230	469	0.49	3.57
31623	59293266	107	264	455	0.79	2.29
100000	104617752	368	975	3457	1.16	4.56
100000	105390777	368	947	3467	0.85	7.47
316228	186870839	1372	3495	32018	0.98	9.44

Table 2. Results for clustered (C) DIMACS instances. Same information than for Table 1. However LKH was run with additional parameter INITIAL_PERIOD = 1000 for speeding-up the preprocessing time.

running it on sub-graphs obtained by the union of 20 POPMUSIC solutions. It is however difficult to compare the quality of both sub-graphs since LKH can produce solutions with edges not present in the sub-graph obtained with 1-trees. LKH also works on distances modified by the 1-trees preprocessing. We see in this table that the computational time is much lower with the POPMUSIC technique for large problem instances.

For clustered instances (Table 2) the sub-graphs obtained with the union of 20 POPMUSIC solutions are clearly better than those obtained with 1-trees. This is visualized in Figure 15.

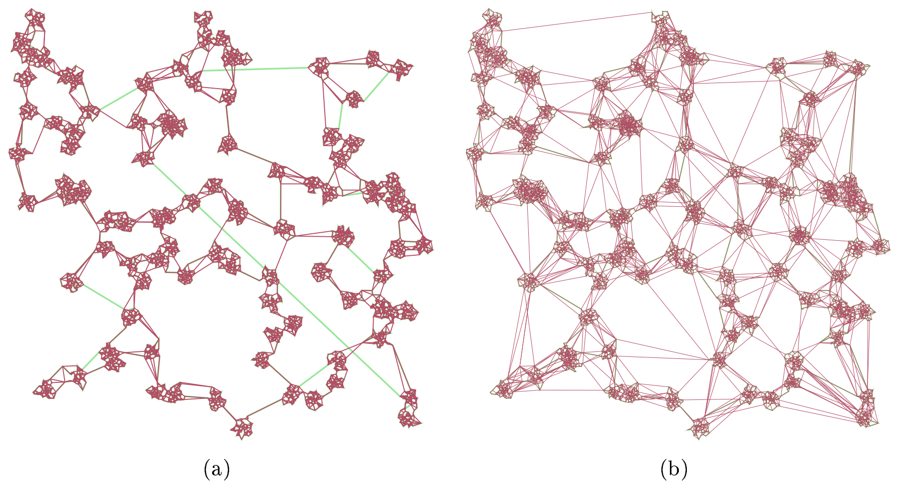

FIGURE 15. Candidate set and solution obtained with LKH on problem instance C10k.2 15(a). The candidate set is not Hamiltonian for this instance, leading to a final solution of poor quality. Union of 20 solutions found by POPMUSIC 15(b) and solution obtained by LKH on this candidate set.

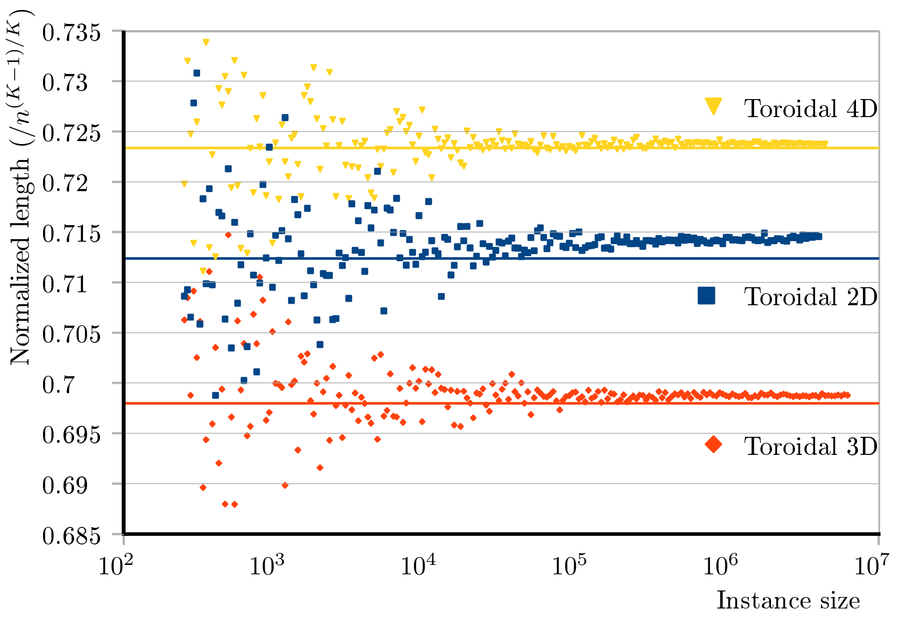

FIGURE 16. Normalized length of solutions obtained by the proposed method as a function of instance size, for problem uniformly, randomly generated in the unit hypercube with toroidal distances in 2D, 3D and 4D. Only one instance of each size is considered to show the dispersion of the length. The horizontal lines show the asymptotic optimal length found by [17].

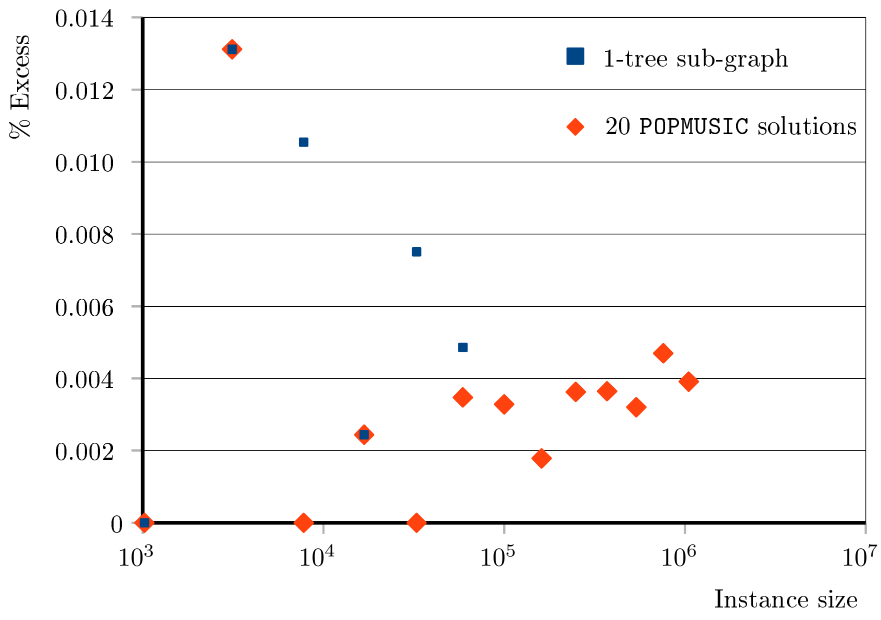

FIGURE 17. Quality of solution produced by LKH with standard parameters (1-trees, 60 trials, 1 run) and the proposed method for regular grids in 5D.

The quality of the solutions produced by the proposed method was also evaluated for uniformly generated problem instances in hypercubes of dimension 2, 3 and 4 with toroidal distances. Figure 16 provides the normalized length of the solutions obtained by the proposed method as a function of instance size. For such instances in dimension K, the normalized length is the length divided by $n^{(K-1)/K}$. [17] have estimated the optimal normalized lengths to 0.7124 ± 0.0002 for $K = 2$, 0.698 ± 0.0003 for $K = 3$ and 0.7234 ± 0.0003 for $K = 4$. For problem instances ranging from 250 to more than 3 million cities, we have obtained average normalized length of 0.7136, 0.699 and 0.7235 respectively (meaning, solution length 0.17% , 0.15% and 0.011% above the predicted optimum).

Figure 16 shows that the quality of the solutions produced does not decrease significantly with instance size. For problem instances with more than half a million cities, the solution length is, respectively, 0.27% , 0.11% and 0.055% above the predicted optimum.

For problem instances on a regular grid in dimension 5, the gap to the optimum solution value is given in Figure 17 as a function of instance size. The solutions obtained are again at a fraction of % above the optimum. The method proposed seems slightly better than LKH with standard parameters.

6. CONCLUSIONS

We have proposed a new general method for the travelling salesman problem with a low empirical complexity, typically in $O(n^{1.6})$. The method has generated solutions of excellent quality (generally a fraction of % above the best solution known) to all problem instances tested, whose size goes up to 10 million cities. The method can be applied to any TSP instance for which the distance between 2 cities can be computed in constant time. It does not make any assumption about the problem structure.

The method could be integrated in software like LKH or Concorde as a fast, general preprocessing option. The general preprocessing based on 1-trees embedded in LKH has an empirical complexity that seems to be quadratic and cannot be used practically for instances with more than 10^5 cities. Moreover, the general preprocessing based on 1-trees may produce sub-graphs of poor quality for non-uniform problem instances. Faster and better preprocessing exists in leading software, but can only be applied to geometrical problem instances.

Future work could be to try other techniques for reducing the complexity of the method, for instance by implementing a recursive decomposition mechanism. The method should also be tested on different problem instances of large size, especially on non geometrical ones.

Acknowledgement

This research was partially supported by the Swiss National Science Foundation, project 200021_169085. We thank the anonymous reviewers for their careful reading of our manuscript and their insightful comments and suggestions.

References

[1] Adriana C. F. Alvim and Éric D. Taillard. POPMUSIC for the point feature label placement problem. *European Journal of Operational Research*, 192(2):396–413, 2009.

[2] Adriana C. F. Alvim and Éric D. Taillard. POPMUSIC for the world location routing problem. *EURO Journal on Transportation and Logistics*, 2(3):231–254, 2013.

[3] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. Concorde: A code for solving traveling salesman problems, 1999.

[4] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, USA, 2007.

[5] David L. Applegate, William Cook, and André Rohe. Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[7] Andrius Blazinskas and Alfonsas Misevicius. Generating high quality candidate sets by tour merging for the traveling salesman problem. In Tomas Skersys, Rimantas Butleris, and Rita Butkiene, editors, *Information and Software Technologies: 18th International Conference Proceedings, ICIST 2012, Kaunas, Lithuania, September 13-14, 2012*, pages 62–73. Springer, Berlin, Heidelberg, 2012.

[8] Jill Cirasella, David S. Johnson, Lyle A. McGeoch, and Weixiong Zhang. The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators, and Tests. In *Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Washington, DC, USA, January 5-6, 2001, Revised Papers*, pages 32–59, 2001.

[9] Wiliam J. Cook and Paul Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[10] William J. Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2012.

[11] Boris Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles*, 6:793–800, 1934.

[12] Stefan Helber and Sahling Florian. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. *International Journal of Production Economics*, 123:247–256, 2010.

[13] Keld Helsgaun. Best known solutions to Dimacs TSP instances. Last updated: October 6, 2014.

[14] Keld Helsgaun. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1, 2009.

[15] Keld Helsgaun. Helsgaun's implementation of Lin-Kernighan, 2016. Version LKH-2.0.7.

[16] David S. Johnson, Lyle A. McGeoch, Fred Glover, and Cesar Régo. 8th dimacs implementation challenge: The traveling salesman problem. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, USA, 2000.

[17] David S. Johnson, Lyle A. McGeoch, and Edward E. Rothberg. Asymptotic experimental analysis for the Held-Karp traveling salesman bound. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 341–350, 1996.

[18] Shen Lin and Brian Wilson Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[19] Peter Merz and Jutta Huhse. An iterated local search approach for finding provably good solutions for very large tsp instances. In Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, pages 929–939, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[20] Colin Osterman and César Rego. Satellite list and new data structures for symmetric traveling salesman problems. Technical report, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, University, MS 38677, USA, 2003.

[21] Alexander Ostertag, Karl F. Doerner, Richard F. Hartl, Éric D. Taillard, and Philippe Waelti. POPMUSIC for a real-world large-scale vehicle routing problem with time windows. *JORS*, 60(7):934–943, 2009.

[22] Giuseppe Paeno. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, March 1890.

[23] David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 399–419. Springer US, Boston, MA, 2010.

[24] Abraham P. Punnen and Gregory Gutin. *The Traveling Salesman Problem and Its Variations.* Combinatorial Optimization 12. Springer US, 1 edition, 2007.

[25] César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427 – 441, 2011.

[26] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *4th International Conference of Principles and Practice of Constraint Programming*, pages 417–431. Springer-Verlag, 1998.

[27] Éric D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673, 1993.

[28] Éric D. Taillard. Heuristic methods for large centroid clustering problems. *J. Heuristics*, 9(1):51–73, 2003.

[29] Éric D. Taillard and Stefan Voss. POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In Celso C. Ribeiro and Pierre Hansen, editors, *Essays and surveys in Metaheuristics*, pages 613–629. Kluwer Academic Publishers, 2001.

[30] Voigt, editor. *Der Handlungsreisende wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein ; Mit einem Titelkupf.* Voigt, Ilmenau, Germany, 1832.

(Éric D. Taillard) HEIG-VD, Department of Industrial Systems, University of Applied Sciences of Western Switzerland, Route de Cheseaux 1, Case postale 521, CH-1401 Yverdon, Switzerland.
E-mail address: `eric.taillard(at)heig-vd.ch`

(Keld Helsgaun) Department of Computer Science, Roskilde University, DK-4000 Roskilde, Denmark
E-mail address: `keld(at)ruc.dk`