A LINEARITHMIC HEURISTIC FOR THE TRAVELLING SALESMAN PROBLEM

ÉRIC D. TAILLARD

Abstract. A linearithmic ($n \log n$) randomized method based on POPMUSIC (Partial Optimization Metaheuristic Under Special Intensification Conditions) is proposed for generating reasonably good solutions to the travelling salesman problem. The method improves a previous work with empirical algorithmic complexity in $n^{1.6}$. The method has been tested on instances with billions of cities. For a lot of problem instances of the literature, a few dozens of runs are able to generate a very high proportion of the edges of the best solutions known. This characteristic is exploited in a new release of the Helsgaun's implementation of the Lin-Kernighan heuristic (LKH) that is also able to produce rapidly extremely good solutions for non-Euclidean instances. The practical limits of the proposed method are discussed on a new type of problem instances arising in a manufacturing process, especially in 3D extrusion printing.

1. Introduction

The travelling salesman problem (TSP) and its variations is probably the most studied NP-hard combinatorial optimisation problem (Voigt, 1832; Punnen & Gutin, 2007; Cook, 2012; Cacchiani, Contreras-Bolton, & Toth, 2020; Ha, Deville, Pham, & Hà, 2020; Campbell, Corberán, Plana, Sanchis, & Segura, 2021). Now, we are able to exactly solve instances up to several thousands of cities and to find solutions for instances with millions of cities at a fraction of a percent above the optimum (Applegate, Bixby, Chvátal, & Cook, 1999; Applegate, Cook, & Rohe, 2003; Merz & Huhse, 2008; Helsgaun, 2009, 2016; Rego, Gamboa, Glover, & Osterman, 2011).

The work of Johnson and McGeoch (1997) has shown that the best heuristic methods are local searches based on a neighbourhood proposed by Lin and Kernighan (1973) (LK for short). This neighbourhood is based on a chain of 2-opt moves, where two edges are replaced by two others. This neighbourhood can be improved by considering k-opt moves, with k up to 5 (Helsgaun, 2009, 2016).

A key point for implementing a fast and efficient local search is to use a neighbourhood of limited size containing the pertinent moves (i.e. moves involving only the edges of fairly good tours). Doing so, only a subset of moves is evaluated for speeding-up the computation. A way to limit the number of moves, is to generate a few dozens of different TSP solutions of moderately good quality with a fast randomized heuristic. Only the edges contained in

Key words and phrases. Travelling Salesman, Local search, POPMUSIC, Large-Scale Optimization, Metaheuristics, 3D printing.
Accepted for publication in EJOR, May 20. 2021.

these tours are considered for building moves. This technique for reducing the complexity of the local search was called *tour merging* by Applegate, Bixby, Chvátal, and Cook (2007). This article proposes a method with a reduced algorithmic complexity for generating a limited subset of pertinent edges that must be used in the solution tour. Previous works (Taillard, 2017; Taillard & Helsgaun, 2019) proposed a 1-level decomposition method for generating good candidate edges. The empirical complexity of the method was approximated with a function proportional to n^{α}. The best estimate of α, after one coma position, was found to be $\alpha \approx 1.6$.

The method proposed in this article is based on two main steps: first, an initial tour is built with a recursive randomized procedure, with a linearithmic algorithmic complexity ($n\ log\ n$). This initial tour is then improved in linear time with a fast POPMUSIC (Partial Optimisation Metaheuristic Under Special Intensification Conditions). Since the complexity of the method is linearithmic, extremely large instances can be treated with a standard personal computer. Experiments with more than 2 billion of cities are reported for the first time. These massive datasets instances consider toroidal distances in 2D. For such instances, the expected optimal solution length is known (Johnson, McGeoch, & Rothberg, 1996). The deviation to the optimal solution observed by the method proposed is about 10%, which is less than the half of the deviation observed by the well-known nearest neighbour greedy heuristics.

These positive results may be explained by the fact that a polynomial approximation scheme exists for geometric instances (Arora, 1998). Very shortly, the polynomial approximation scheme suggested by Arora (1998) proceeds by decomposing TSP instances into a large number of "independent" and smaller instances. The algorithm proposed in this article also exploits instance decomposition.

To explore the limits of the proposed method, the last is tested on TSP instances for which the distances are not purely geometric. First, instances of clustered TSP (CTSP) have been transformed into standard TSP instances by the technique consisting of adding an arbitrarily large constant M to inter-cluster costs. It is known that the technique introduce a fair amount of degeneracy in the problem (Laporte & Palekar, 2002). Despite this, 20 runs of the method proposed in this article are able to find more than 99% of the edges contained in the best solutions known for a classical CTSP data set. The same number of runs of the nearest neighbour heuristic is able to find less than 86% of these edges.

To build instances with a larger amount of degeneracy, a new application occurring in 3D extrusion printing is proposed. It is shown that modelling the problem of minimizing the unproductive moves of the extrusion head as a TSP and solving this TSP with a state-of-the-art method (Helsgaun, 2018) can vastly improve the solutions reported in the literature (Volpato, Galvão, Nunes, Souza, & Oguido, 2020). However, we have observed that the proposed method cannot produce a reasonable solution to a 3D extrusion printing instance with several hundred of layers expressed as a single TSP instance with several hundred-thousands of cities.

The article is structured as follows: Section 2 presents the new *n log n* heuristic for the TSP. Section 3 presents a first analysis of the method on standard TSP. First, it is shown that the empirical complexity of the method is linearithmic, as expected. Then the solution quality produced for very large toroidal instances is analysed. Finally, it is shown that the method generates a very large proportion of edges belonging to the best solution tours. Section 4 introduces a TSP model for minimizing the unproductive moves in production processes. Numerical results show that the model can efficiently solve this kind of problem with a good TSP solver. These instances are also used to explain the limits of the new heuristic presented in this paper. Conclusions and future research avenues are presented in the Section 5.

2. The linearithmic Heuristic for the TSP

The new heuristic proposed for finding a tour on a set C of n cities is decomposed in two phases. The first phase builds an initial path, including all the cities of the instance by a recursive process. Then, in a second phase, the initial tour is improved by a truncated POPMUSIC algorithm. The Algorithm 1 gives a sketch of the new algorithm proposed. It calls two procedures that are presented in the next sub-sections. These procedures require both another procedure that is able to provide a very good path that includes all the cities of a relatively small subset of cities. This procedure can be either a local search based on LK neighbourhood, or a local search based on a reduced set of 3-opt moves (this is the choice in the LKH 2.0.9 implementation (Helsgaun, 2018)) or any other optimisation procedure, even an exact one, provided that it can deal with few hundreds of cities.

Algorithm 1: Overview of the fast heuristic proposed for the TSP. Procedures *ReorderPath* and *FastPopmusic* are discussed in the next sub-sections.

Data: Set C of n cities $c_1, \ldots, c_n$, parameter $t \leqslant n$
Result: Tour T including all cities of C
1 Choose a random city $c_i \in C$;
2 $P = \mathit{ReorderPath}((c_i, \ldots, c_{i-1}, c_{i+1}, \ldots, c_i), t)$;
3 $T = \mathit{FastPopmusic}(P, t^2)$;

For adapting POPMUSIC to the TSP, the choice was to optimise sub-paths of an initial tour. One of the "special conditions" required by this technique is that two portions of the initial tour, separated by many cities, should not be interleaved. So, an appropriate procedure must be designed for generating an initial solution meeting this condition.

2.1. Building an initial tour.

For generating an initial tour that can be further improved with a fast POPMUSIC algorithm, a city c_i is randomly chosen. Since the method is based on improving paths, a feasible TSP tour can be seen as finding a path beginning in c_i and ending in c_i. So, the method starts with the path $P = (b = c_i, c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n, c_i = e)$ that defines a (random) feasible TSP tour.

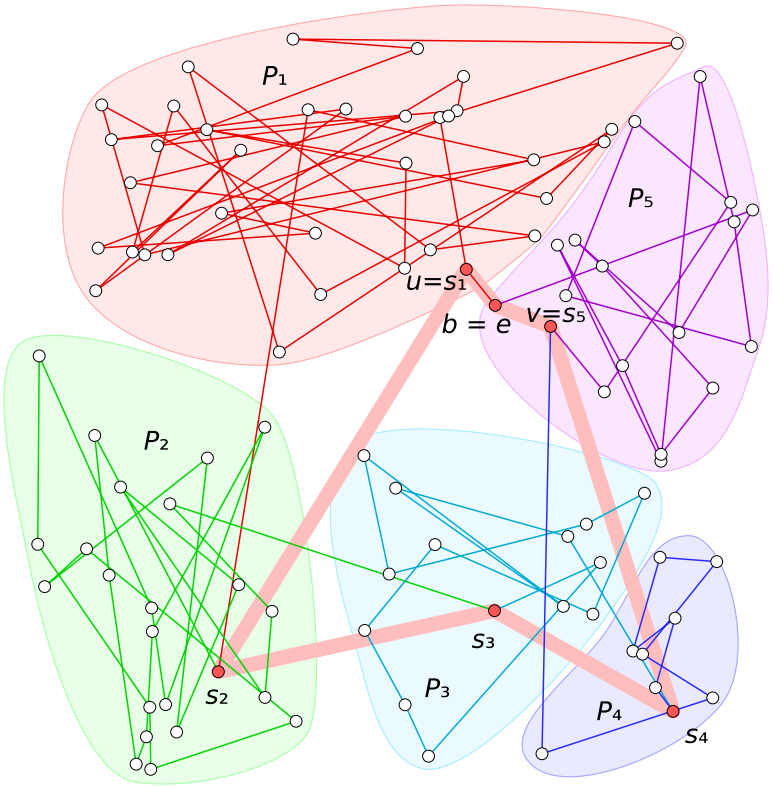

FIGURE 1. Illustration of Algorithm 2 on a small instance (first 100 cities of Dimacs E1k.0). First call of the procedure, with parameter $t = 5$. Path on a random sample (bold and light line) and reordered path P_S completed with all cities before recursive calls of the procedure at line 12. Paths P_1 to P_5 are drawn with different colours and different backgrounds highlight them.

Let t be the only parameter of the method, not depending on n. A shown in Section 3, a value of t between 10 and 20 is convenient. If not specified otherwise, the LKH 2.0.9 implementation takes $t = 10$.

If $n \leq t^2$, then a very good path passing once through all cities of P, beginning at city b and ending at city e can be found, for instance with a local search using LK moves or even with an exact method. A good tour is built and the method stops.

Else, if $n > t^2$, a sample S of t cities is chosen by including in S:

- $u \in C \setminus \{b, e\}$, the city closest to b
- $v \in C \setminus \{b, e, u\}$, the city closest to e
- $t - 2$ other cities of $C \setminus \{b, e, u, v\}$ randomly picked

A good path P_S through all the cities of sample S, starting at city b and ending at city e can be found with a local search or an exact method. Let us rename the cities of S so that $P_S = b, s_1, s_2, \ldots, s_{t-1}, s_t, e$.

Path P_S can be completed to contain all the cities of C by inserting them, one after the others, just after the closest city of S. So, the completed path

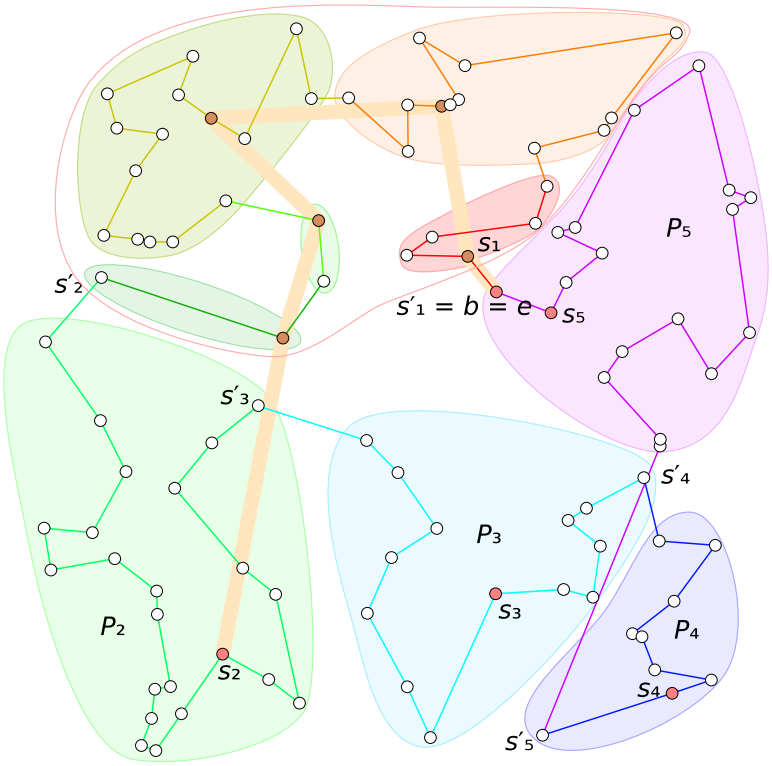

FIGURE 2. Illustration of Algorithm 2. A final path that can be returned by the algorithm on the instance of Figure 1. The path P_1 was recursively decomposed into $t = 5$ pieces. The path on a sample of cities of P_1 (plus the city preceding P_1 and the city s_2 of P_2) found at line 7 of the algorithm is indicated by the bold and light line. The paths P_2 to P_5 contain no more than $t^2 = 25$ cities and are directly optimised.

$P_S = (b, s_1, \ldots, s_2, \ldots, s_t, \ldots, e)$ improves the initial path P. Figure 1 gives an example of P_S.

At this step, the order of the cities in the completed path P_S between two cities s_j and $s_j + 1$ is arbitrary (as it was for P at the beginning of the procedure). The sub-paths $P_1 = (b = s'_1, \ldots, s_2), P_2 = (s'_2, \ldots, s_3), \ldots, P_t = (s'_t, \ldots, e = s_{t+1}) \subset P_S$ can be further improved with t recursive calls of the same procedure, where s'_j is the city just preceding the first one of the path P_j. The Algorithm 2 describes the process in details. Figure 2 gives the final tour found by applying Algorithm 2 for a small instance.

When all recursions stop, the cities are ordered in such a way that the TSP tour can be successfully improved with a POPMUSIC-based heuristic. If t is considered as a fixed parameter (not depending on the problem size n), the algorithmic complexity of this procedure is $n \log n$.

2.2. **Improving the initial tour with a fast POPMUSIC.** In Taillard (2017), POPMUSIC metaheuristic is adapted for the TSP by optimizing

Algorithm 2: *ReorderPath*. Procedure to improve a path P. It requires a procedure *OptimisePath* able to efficiently find a good path for instances containing less than t^2 cities.

Data: Path $P = (b, \ldots, e)$, parameter t
Result: Path P_S from b to e containing all cities of P reordered
if $|P| \leqslant t^2$ **then**
  $P_S = OptimisePath(P)$
else
  Find $u \in P \setminus \{b, e\}$, the closest city to b;
  Find $v \in P \setminus \{b, e, u\}$, the closest city to e;
  Randomly select a sample S of $t-2$ cities from $P \setminus \{b, e, u, v\}$
  $S \leftarrow S \cup \{u, v\}$;
  $P_S = (b, s_1, \ldots, s_t, e) = OptimisePath((b, (S), e))$;
  for $c_j \in P, c_j \notin P_S$ **do**
    Find the city $s \in S$ the closest to c_j;
    Insert c_j in P_S just after s
  $s_{t+1} = e$;
  for $j \in \{1, \ldots, t\}$ **do**
    s'_j = city preceding s_j in P_S;
    P_j = sub-path of P_S from s'_j to s_{j+1};
    $P_j \leftarrow ReorderPath(P_j, t)$;
    In P_S, replace the sub-path from s'_j to s_{j+1} by P_j;

sub-paths containing R consecutive cities of the tour, where R is a parameter. The optimization procedure can be exactly the same as those used in Algorithm 2. The optimizations are repeated until there is no subset of R consecutive cities in the tour that can be improved.

For getting good candidate edges by the tour merging technique, we have remarked that it is not required to run POPMUSIC until all sub-paths of R consecutive cities have been optimised. Instead, we propose to speed up the method by optimising the tour in 2 scans. A first scan optimizes $\lceil n/R \rceil$ non-overlapping sub-paths of R cities at most. Then, the tour is shifted by $\lfloor R/2 \rfloor$ cities and a second scan optimises sub-paths involving about $R/2$ cities of two adjacent sub-paths of the first scan.

If R does not depend on n, then the algorithmic complexity of the improvement with the fast POPMUSIC is linear with n. To have a method with a unique parameter, R can be set to t^2. Figure 3 illustrates the improvement of a solution with the fast POPMUSIC technique.

3. Computational results on classical TSP instances

The heuristic method proposed in this article was coded in C. The programming style was purely sequential, so only one CPU core is used during the runs. The personal computer used for the numerical experiments had an AMD 1950X processor, 64GB of memory and operates on Linux (Xubuntu).

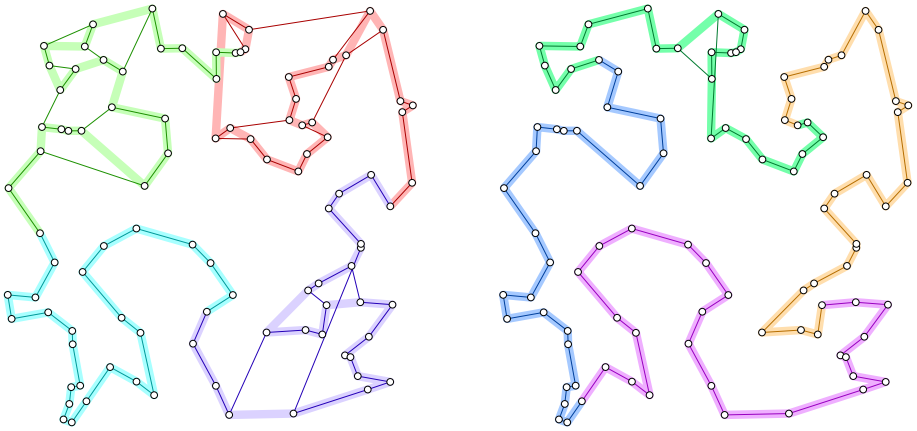

FIGURE 3. Improvement of the tour obtained with Algorithm 2 with the fast POPMUSIC. Left: first scan of four independent paths (connected by their extremities only); right: second scan shifted by 12 cities. Thin lines are the initial paths and bold, light lines are the optimised paths.

The compiler was *gcc* and the code was compiled with -O4 optimisation option.

3.1. **Parameter setting.** Intrinsically, the method has three parameters: the maximum number of cities in a path $path_{max}$ for a direct optimisation of the paths (Line 1 of Algorithm 2), the sample size t (Line 6 of Algorithm 2) and the maximal number of cities R of the sub-paths optimised with the fast POPMUSIC. Moreover, Algorithm 2 requires a procedure for optimising a path between two fixed cities. For all the numerical results presented in this paper, a local search based on LK moves has been used. This procedure is a very basic one, implemented in less than 100 lines of C code. The same procedure was used in our previous work (Taillard & Helsgaun, 2019).

Preliminary results have shown that the method is most sensitive to parameter R. The quality of the solutions produced increases with R, and so do the computational time. The main goal of this work was to produce moderately good solutions to TSP instances with a short computational time. So, R should be set to a relatively small value. However, it is not pertinent to set R to a value (much) smaller than $path_{max}$. Indeed, there might be portions of the tour produced with Algorithm 2 that have already been optimised with $path_{max}$ cities. Our choice was to set $R = path_{max}$ so that the improvement of the tour with the fast POPMUSIC method is not marginal.

A path containing more than $path_{max}$ cities is split into t portions. On the average, each portion will contain $path_{max}/t$ cities. The portions should not be too small for producing pertinent edges, once optimised. Our choice was to set $path_{max} = t^2$. The above considerations lead us to propose a method with a single parameter t, implicitly fixing the value of the other parameters to t^2.

The method is first tested for empirically evaluating the relation between the value t of the parameter, the instance size, the solution quality and

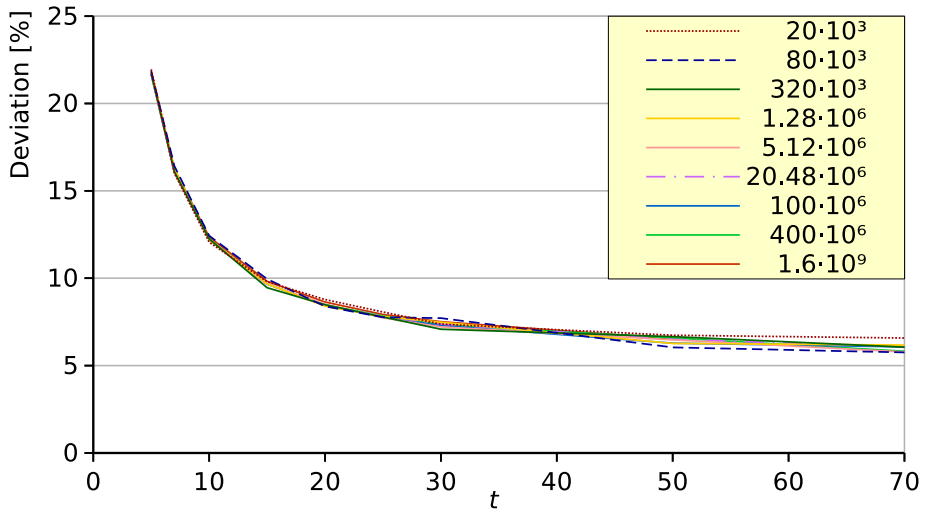

FIGURE 4. Quality of the solutions (expressed in % above expected optimum) produced by the method as a function of parameter t for 2D toroidal instances. The size of the instances varies from $2 \cdot 10^4$ to $1.6 \cdot 10^9$ cities.

the computational time. The largest problem instances commonly available have a few millions of cities. To test the method on larger instances, we have generated instances randomly on a unit square and used toroidal distances (as if the square was folded so that opposite borders are contiguous). As the asymptotic optimal length is known for such instances, it is possible to evaluate the quality of the solutions produced by our method too.

Figure 4 gives the quality of the solutions produced by the method as a function of parameter t for 2D toroidal uniform instances. We see in this figure that the deviation to optimum decreases quite fast from $t = 5$ to $t = 15$ and very slowly for $t > 30$. The quality of the solutions produced is quite insensitive to the problem size.

Figure 5 gives the computational time of the method as a function of parameter t for 2D toroidal uniform instances. We see in this figure that the computational time increases relatively fast with t. A good trade-off between computational time and solution quality is to choose the value of parameter t between 10 and 20.

3.2. **Computational times.** The previous POPMUSIC implementation was able to produce solutions to instances with 10^7 cities in about a couple of hours of computational time (Taillard & Helsgaun, 2019). Most of the effort for instances of this size was due to the building of the initial solution and not for its further improvement with POPMUSIC. The previous implementation used a 1-level decomposition. The size of the sample for decomposing the problem depended on the instance size n. The dependence was a fractional power carefully chosen according to the empirical complexity of the local search so that the resulting complexity of the whole procedure is minimised. The observed complexity of this previous implementation was about $n^{1.6}$.

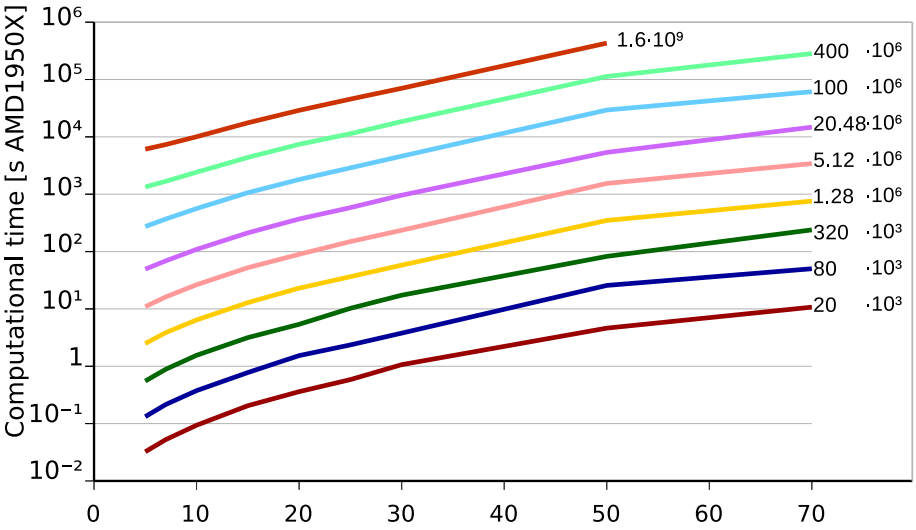

FIGURE 5. Computational time of the method as a function of parameter t for 2D toroidal instances. The size of the instances varies from $2 \cdot 10^4$ to $1.6 \cdot 10^9$ cities.

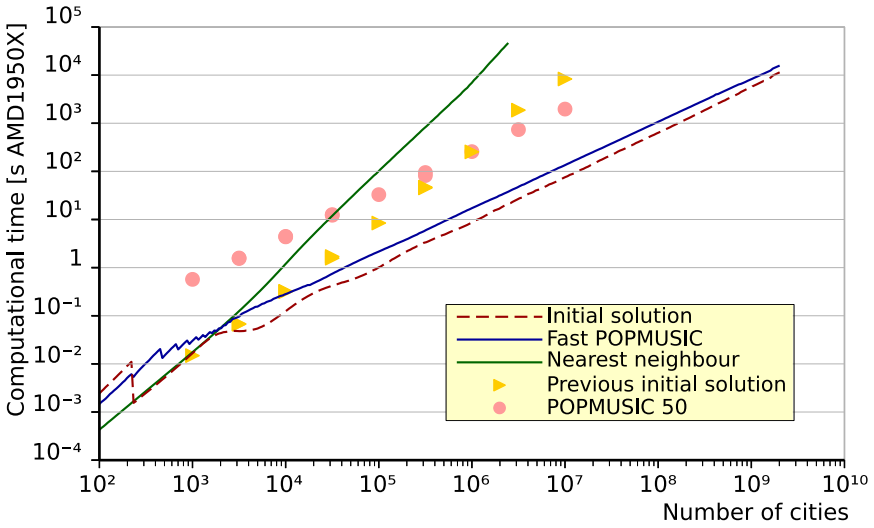

FIGURE 6. Computational time for producing one solution as a function of problem size. The proposed method is run with $t = 15$.

In the present work, the sample size does not depend on the instance size. Therefore, a multi-level ($log\ n$) decomposition is needed. Another difference is that POPMUSIC is truncated. There is no guarantee any more that any sub-path of R cities is locally optimal. The time spent for the improvement of the initial tour is still proportional to n as it was for the previous implementation, but it is divided by a factor of 10 to 20.

Figure 6, provides the computational time of our method as a function of the problem size for 2D toroidal instances. We provide the average time for generating the initial solution and for improving it with the fast POPMUSIC

with parameter $t = 15$. So, the total time of our method is the sum of these times. This figure also includes the computational time for generating a tour and its improvement with the previous POPMUSIC implementation as well as the time for the nearest neighbour greedy procedure. For the nearest neighbour procedure and the fast POPMUSIC, $\min(1000, \lfloor \frac{2^{31}}{n^{1.5}} + 1 \rfloor)$ instances were considered for each number n of cities. This figure shows that:

- the empirical time for generating an initial solution is asymptotically linearithmic, as predicted; the oscillations that can be observed for small problem instances is due to transitions in recursion level; the first, very clear transition occurs for $t^2 = 225$ (from a direct optimisation of the tour to a 1-level decomposition);
- the empirical time for improving the initial solution with an accelerated POPMUSIC is asymptotically linear; the non-monotonic increase that can be observed for small instances is due to changes in the number of cities in the sub-paths optimisation; for instance, up to $n = t^2$ there are two calls of the procedure for improving paths with n cities and for $n = t^2 + 1$, there are four calls of this procedure for improving paths with $n/2$ cities;
- the previous POPMUSIC implementation, optimising sub-paths of 50 cities, is more than 10 times slower that the accelerated POPMUSIC;
- the empirical algorithmic complexity of the nearest neighbour heuristic is very close to the $\Theta(n^2)$ theoretical complexity.

The main limitation of our method is due to the memory required for storing the problem data and the solution. Indeed, 2×8 bytes are required for storing the coordinates of a city (meaning 32GB for $2^{31} \approx 2 \cdot 10^9$ cities) and a solution requires an array of 4 byte integers. In our implementation, a temporary copy of the solution is needed as well as an array for storing the assignment to the nearest city of the sample. So, at the first decomposition level, 32GB + $2^{31} \cdot 3 \cdot 4$B = 56GB are required for a 2 billion cities instance. Since a "small" amount of memory is required for the recursive calls, we can conclude that the RAM of our personal computer was almost exhausted — however, without requiring memory swaps on the hard disk.

3.3. **Solution quality.** The optimal solution length of randomly generated instances in the unit square with toroidal distances in dimension D was estimated as $\gamma_D \cdot n^{\frac{D-1}{D}}$, with $\gamma_2 = 0.7124 \pm 0.0002, \gamma_3 = 0.698 \pm 0.0003$ and $\gamma_4 = 0.7234 \pm 0.0003$ by Johnson et al. (1996).

For instances with $D = 2$, the quality of the solutions produced by the nearest neighbour greedy procedure, by our method with $t = 15$ and by a 1-level recursion method with running time similar to our previous POPMUSIC implementation (setting $t = R = 1.5 \cdot \sqrt{n}$ and $max_{path} = n - 1$) is given in Figure 7 .

This figure shows that:

- for a fixed value of t, the quality of the initial solution fluctuates around 16% above predicted optimum, with peaks near power of t;

		Algorithm 2		Algorithm 1		
D	t	$n = t^2 + 1$	$n \gg t^2$	$n \leqslant t^2$	$n = t^2 + 1$	$n \gg t^2$
2	10	23.4	19.7	3.4	12.7	12.3
2	15	20.8	15.7	3.6	11.1	9.7
2	20	19.2	13.5	3.7	9.9	8.5
2	25	17.8	12.3	3.7	9.1	7.8
2	30	16.8	11.4	3.8	8.7	7.3
3	15	23.0	19.1	2.3	10.5	12.3
3	25	20.3	14.6	2.4	8.9	9.1
4	15	23.8	22.9	1.6	9.5	15.8
4	25	22.1	18.2	1.7	8.7	11.6

TABLE 1. Quality produced for different parameter t and different dimensions of toroidal instances, expressed in % above predicted optimum. For $n \leqslant t^2$, the instance is directly solved with a local search based on LK neighbourhood.

- for a value of t proportional to $\sqrt{n}$, the quality of the solution increases with n;
- the deviation over expected optimum is quite stable after the improvement of the initial solution by the accelerated POPMUSIC; this deviation stays between 8% and 11% over the expected optimum (a quality similar to our previous POPMUSIC implementation);
- the nearest neighbour procedure produces solutions at $21.96 \pm 0.3\%$ above predicted optimum for problem instances above 30,000 cities.

Table 1 provides the average solution quality for toroidal instances. The first column gives the dimension D of the instances. The second column gives the value of parameter t. The next two columns gives the quality provided by the Algorithm 2 (initial solution). On the one hand is provided the results for the worst case for this algorithm, when n is just above t^2. On the other hand is given the asymptotic behaviour observed for very large sizes. The last tree columns gives the results of the proposed method. First is given the average deviation over expected optimum observed for instances of size varying between $n = 100$ and $n = t^2$. For such sizes, no recursive call occurs in Algorithm 2 and the results correspond to the quality provided by the *OptimisePath* procedure. Then are the results when the proposed method has a single recursive call ($n = t^2 + 1$) and finally the asymptotic results for very large sizes.

3.4. **Generating candidate edges with tour merging.** The main purpose of the proposed method is for producing good candidate edges for a more elaborated local search. It is now time to analyse the ability of the method to reach this goal. For this purpose, a few classical instances of the literature for which excellent solutions are known have been considered. The size of these instances ranges from 10^4 to $744,710$.

In Figure 8, we give, as a function of the number of runs of our method, the proportion of edges that are in the best solution known but that have not been produced by our method. We see in this figure that the union of

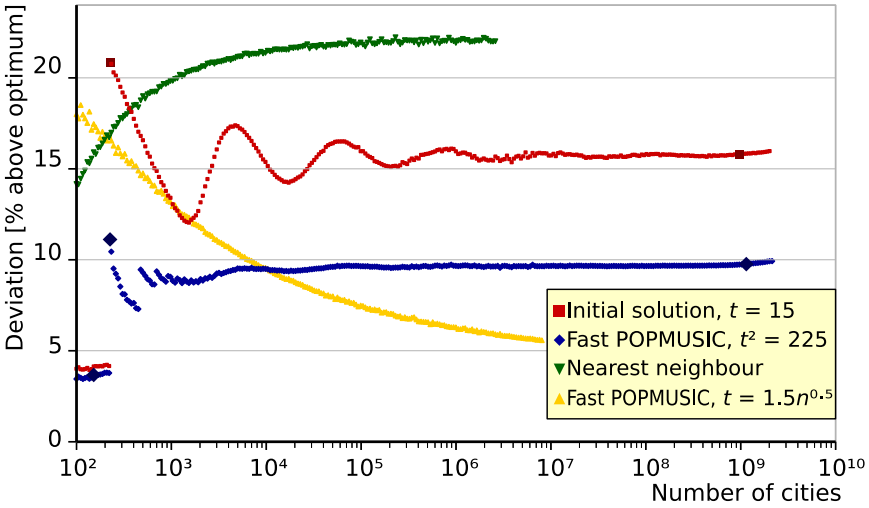

FIGURE 7. Quality of solutions (expressed in % above expected optimum) as a function of problem size. Uniformly distributed cities with toroidal distances in 2D. Larger and darker symbols identify the 5 typical measures given in Table 1 (line with $D = 2$ and $t = 15$).

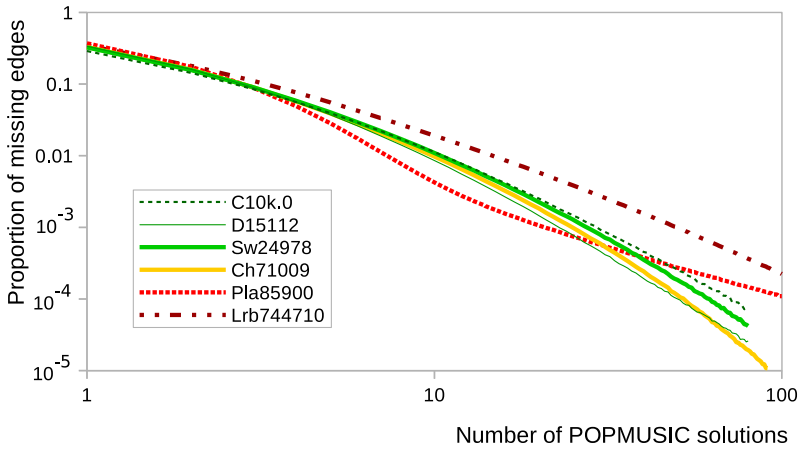

FIGURE 8. Proportion of missing edges as a function of the number of fast POPMUSIC solutions generated (with $t = 15$).

50 solutions produced by the method contains more than 99.9% of the edges of the best solution known.

So, the proposed method is a very efficient one to generate good candidate edges by tour merging. The new algorithm has been integrated in the LKH TSP solver since version 2.0.9. The generation of candidate edges with the method proposed in this article can be enabled with the parameter `CANDIDATE_SET_TYPE = POPMUSIC`. The technical report provides more

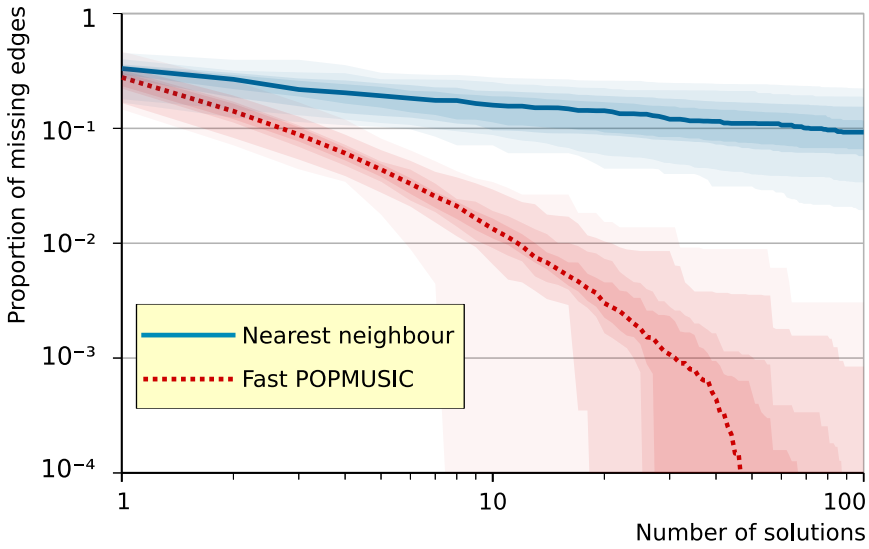

FIGURE 9. Clustered TSP: Median proportion of missing edges as a function of the number of solutions generated. The nearest neighbour heuristic starts from a random city. The shaded zones indicate the 0–100, 5–95, 15–85 and 25–75 percentiles.

details about the solution quality and computational time of LKH TSP software when enabling this parameter (Helsgaun, 2018).

A criticism of these results can be that, although dealing with instances of various sizes, they are all based on geometrical distances.

So, the method must be tested on instances with quite different distance measures. For this purpose, the clustered TSP (CTSP) is considered. In the CTSP, each city is associated with a given cluster. All the cities of a cluster must be visited before going to a city associated with another cluster. The CTSP can be transformed into a standard TSP by adding an arbitrarily large constant M to inter-cluster costs. So, if the true distance between cities i and j is d_{ij}, the distance in the TSP model is $d_{ij} + M$ if i and j belong to different clusters. Adding M at each departure of a cluster implies that an optimal TSP tour must visit all the cities of a cluster before going to a city of another cluster. So, an optimal TSP tour defines a feasible CTSP tour. If the CTSP has m clusters, then the corresponding TSP optimal solution value is mM above the CTSP one.

A classical data set of 65 CTSP instances has been considered. The size of these instances ranges from 200 to 71,009. Very good solutions for these instances are available (Helsgaun, 2014). In Figure 9, we give the proportion of missing edges as a function of the number of runs of our method and the nearest neighbour heuristic starting from a random city. We see in this figure that 20 runs of our method are able to find more than 99% of the edges of the best solutions while 100 runs of nearest neighbour find only 90% of them.

We can conclude that repeated runs of our method is able to generate a very good proportion of edges appearing in excellent CTSP solutions. Let

us mention that the nearest neighbour heuristic always finds a CTSP feasible solution. This contrasts with our method that may produce unfeasible CTSP solutions. For ensuring the production of feasible CTSP solutions, our method should be modified by enforcing the selection of one city from each cluster at the first sampling.

The time complexity of the nearest neighbour heuristic is $\Theta(n^2)$. It is slightly faster than our method for problem instances with $n = 1000$ cities but 30 slower for the largest instance with $n = 71,009$ cities.

4. TSP model for optimising unproductive moves in manufacturing

This section presents a way to model the optimisation of unproductive moves in an additive manufacturing process as a TSP. The problem occurs in the production of 3D pieces with extrusion printers. Figure 10 shows such a piece. The different colors represent different types of extrusion: external surface, filling, etc.

A piece that must be produced with a 3D printer is first sliced horizontally into a number of layers. The printer prints one layer after the other. Once a layer is printed, the head (or the support of the piece) is shifted vertically from a distance corresponding to the thickness of a layer. Then, the next layer is printed.

Practically, a layer is composed by a number of segments that must be extruded. One of these segments is specified by the (x, y) coordinates of its extremities and other parameters such as the print speed, the extrusion quantity, the geometry of the segment (straight line, arc) and the type (contour, filling). A layer may be decomposed into sub-layers, for instance, if all the contour segments must be printed before the filling ones.

The problem for a (sub-) layer is to decide in which order printing each segment and which is the start extremity and the end extremity of each segment. If the coordinates of the end extremity of a segment are not the same as those of the start extremity of the next one, the extrusion head must perform a non-productive jump in the air. The non-productive moves may represent a significant proportion of the head moves. For instance, in Figure 11, the length of non-productive moves of a layer generated by the *PrusaSlicer* software is more than 20% of the length of the productive moves.

For some pieces, the time consumed in non-productive moves can be higher than the time for productive ones. Therefore, it is important to determine an order and a direction for each segment that minimises the time of non-productive moves.

For modelling the minimisation of non-productive time as a TSP, we propose to introduce 3 cities for each segment. Let a, b, c be the cities associated with a segment and e, f, g the cities associated with another segment. The

TSP distance matrix $\mathbf{D}$ between these 6 cities can be defined as

$$\mathbf{D} = \begin{pmatrix} - & 0 & M & d_{ae}+p & M & d_{ag}+p \\ 0 & - & 0 & M & M & M \\ M & 0 & - & d_{ce}+p & M & d_{cg}+p \\ d_{ea}+p & M & d_{ec}+p & - & 0 & M \\ M & M & M & 0 & - & 0 \\ d_{ga}+p & M & d_{gc}+p & M & 0 & - \end{pmatrix}$$

where d_{ij} is the Euclidean distance between the extremity i of a segment and j the extremity of another one, M is a relatively large value (e.g. 100 times the largest distance between 2 points of the printed piece) and p a penalty. Penalty p is equal to:

- 0 if both segments belong to the same layer
- $M/10$ if both segments belong to adjacent layers
- M if both segments belong to non-adjacent layers

To complete the model, a dummy city is added. The dummy city is located at the initial position of the extrusion head. The distance of this dummy city to all the others is M, but excepted for the extremities of the segments of the first and last layer.

Printing a segment means going from one of its extremities to the other. This is ensured in our TSP model for an optimal tour by the fact that the only short connections to the middle city associated with a segment (in the preceding example, the city b or the city f) are precisely the extremities of the segment. All other cities are very far away from the middle city.

A feasible solution for 3D printing must print all the segments of a layer before starting the next layer. This is ensured in our TSP model by the penalty p that has to be paid in addition to the distance between the extremities of two distinct segments. No penalty is paid if both segments belong to the same layer (accounting therefore the unproductive jump distance). A modest penalty is paid if both segments belong to adjacent layers. This corresponds to the inter-cluster penalty when transforming the CTSP into a TSP. This ensures to entirely print a layer before going to the next one. Finally, a large penalty is paid if both segments belong to different layers that are not adjacent. This ensures to print the layers in the right order (or completely in reverse order, since the TSP is symmetric).

To validate this model, we have built a TSP instance of 5 successive layers for the piece shown in Figure 10 and we have found a good TSP solution with LKH 2.0.9. Figure 12 illustrates the movements of the extrusion head for the same layer as Figure 11 with non-productive moves put in evidence. Compared to the *PrusaSlicer* solution, the non-productive moves have been diminished by 89% for this layer. For the whole piece, the non-productive moves can be diminished nearly by 75%.

We have remarked that the TSP instances modelling 3D extrusion printing are relatively harder to solve than instances of classical benchmarks. Indeed, even in small instances with few hundreds of cities, LKH 2.0.9 was unable to provide optimal solutions when running with standard parameter values. However, good solutions can be repeatedly found by increasing the

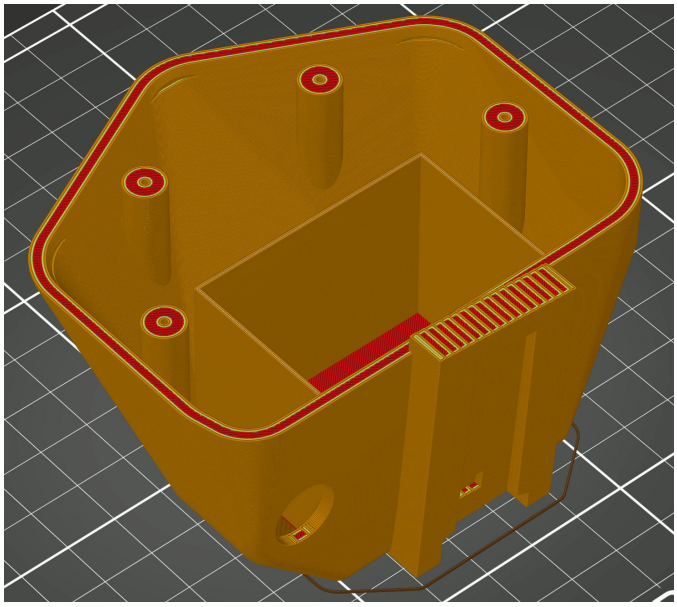

FIGURE 10. Piece of a bicycle headlamp as shown in the *PrusaSlicer* software.

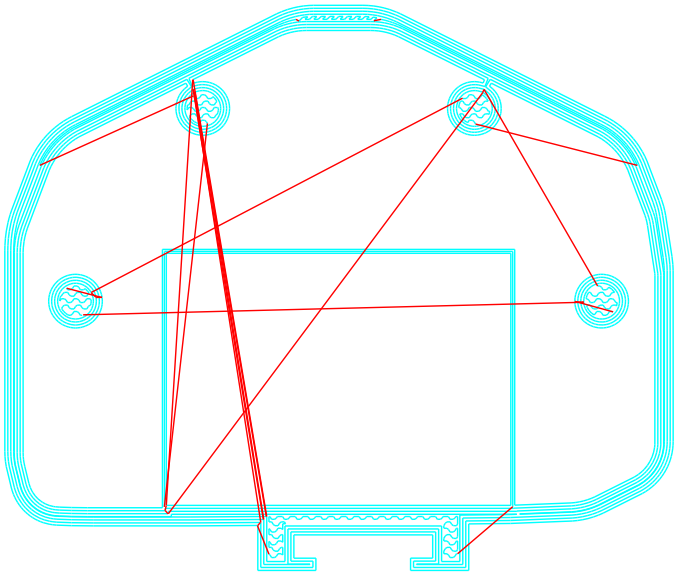

FIGURE 11. A layer of a 3D extrusion printing generated by *PrusaSlicer*. The darker segments are the unproductive jumps. The total length of these jumps for this layer is about 740.8 mm.

number of candidate edges connecting each city from 5 (standard value) to `MAX_CANDIDATES = 10`.

We have tested the LKH TSP solver on 3D extrusion instances proposed by Volpato et al. (2020). This test-bed includes eight different parts. The

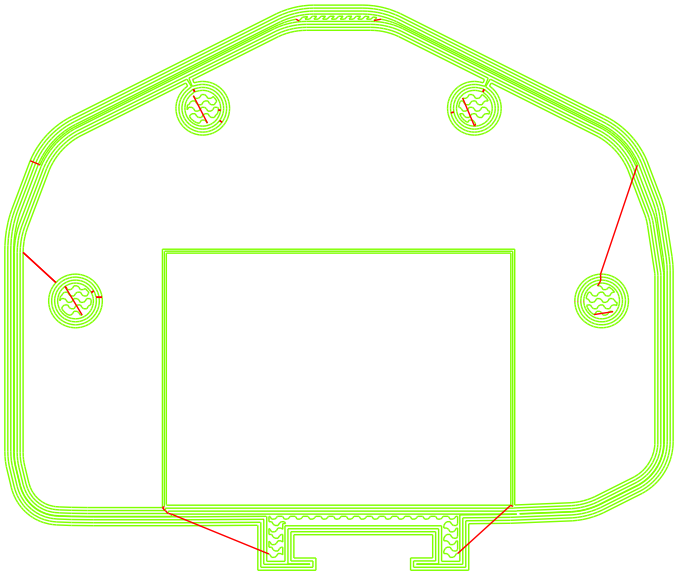

FIGURE 12. A layer of a 3D extrusion printing optimised with a TSP model. The total length of unproductive jumps for this layer is about 84.5 mm.

Instance	1	2	3	4	5	6	7	8
Size	4338	5427	4503	3669	13359	6606	6066	10872
Difference (%)	-0.4	0.6	0.7	0.7	-0.8	0.1	1.4	0.9

TABLE 2. Improvement in unproductive move length of LKH TSP solver when enabling `CANDIDATE_SET_TYPE = POPMUSIC` parameter. Extrusion head cleaned every two layers. Positive values indicate LKH produces better solutions with POPMUSIC candidates.

layers of these instances are decomposed into two sub-layers: contour and raster filling.

For 3D printing instances, the distance data must be given explicitly in LKH TSP solver. So, to limit the size of the instances, we have considered only the instances where the extrusion head must be cleaned every two layers. Since the head is cleaned at a given position, the problem can be decomposed into independent instances containing only two layers. Table 2 provides the relative difference (%) of unproductive length when the LKH TSP solver is run with `CANDIDATE_SET_TYPE = POPMUSIC` parameter. We see in this table that the influence of the candidate edges set selection procedure is not significant for such instances.

The Figure 13 illustrates two layers of a solution obtained by a TSP model. Note that the instance is not provided with the detailed zigzag path for raster filling. The lasts are merely given by a single segment, drawn with a straight line in this figure.

The constraint of printing all the contour segments (close loop) before filling the solid parts is somewhat arbitrary. So, we have generated detailed

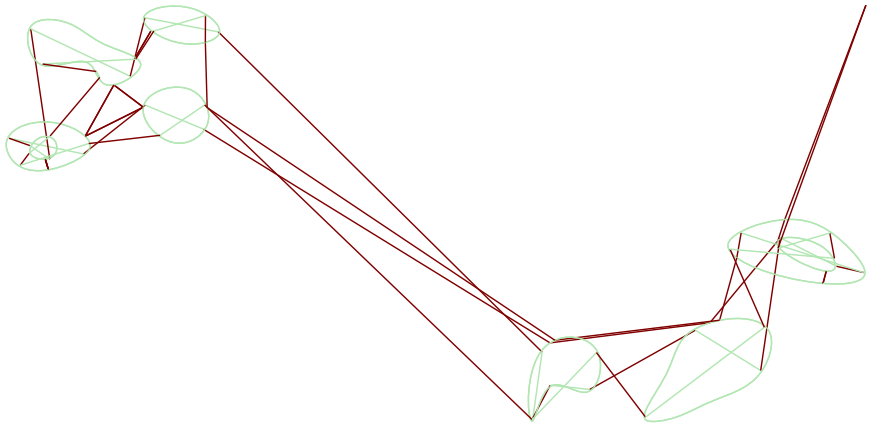

FIGURE 13. Solution found for instance 4 of Volpato et al. with head cleaning every two layers. Darker lines identify non-productive moves.

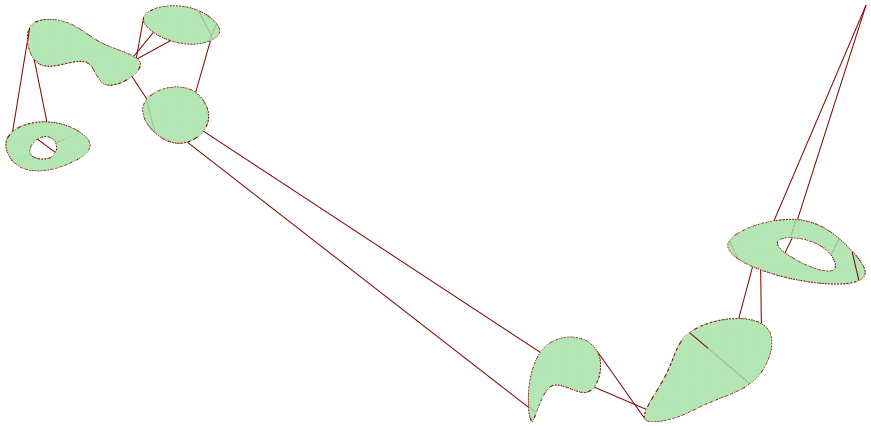

FIGURE 14. Solution found for raster filling of instance 4 of Volpato et al. Head cleaning every two layers. Darker lines identify non-productive moves.

filling segments (crossing at 90° from one layer to the next) for the parts corresponding to the instances of Volpato et al. (2020). The instances are solved by taking into account the raster filling only. The rationale is that the first time the head crosses a contour segment, it can print the contour before printing the raster segment without an increase in the length of non-productive moves. All the raster segments generated for a layer are parallel and disjoints. So, connecting all of them implies non-productive moves whose length is near half the contour length.

The Figure 14 illustrates the solution obtained for the same part as the Figure 13.

Two ad-hoc heuristic methods are proposed in Volpato et al. (2020) for solving the minimisation of unproductive moves in 3D extrusion printing. These methods are shown to improve the repositioning distance of the extrusion head by 43% to 63%. Table 3 reports the reduction in path length

Instance	1	2	3	4	5	6	7	8
Size	2574	2997	3159	3516	6696	4122	6834	6513
Method 1	8.8	30.2	24.0	4.6	38.9	20.8	24.3	11.7
Method 2	6.9	24.6	22.4	1.4	29.7	11.4	15.6	9.6

TABLE 3. Relative improvement of the length of non-productive moves (%) by solving a raster-filling TSP model with LKH TSP solver compared to Method 1 and 2 of Volpato et al. Extrusion head cleaned every two layers. The new TSP model produces better solutions for all instances.

(%) compared to methods 1 and 2 of Volpato et al. (2020). We see that the improvements can be relatively substantial for most of the parts. For Instance 4, the improvement is not that important, which may be surprising while comparing the reposition moves of figures 13 and 14. This is due to the fact that the length connecting two raster segments is taken into account in the raster filling model while it is not taken into account in the contour + zigzag raster filling.

The TSP instances deriving from 3D extrusion printing are pathological for the fast heuristic proposed in this article. Acceptable solutions were obtained when the number of layers is limited, but this is not true with hundreds of layers (and several hundred-thousands of cities). An explanation is that the decomposition process is not accurate when the number of layers is large. Indeed, 1/3 of the cities, corresponding to "middle" of segments, are connected with the same distance, M, to all the other cities but excepted for the extremities of the segment. So, at line 9 of the Algorithm 2, most of the cities are inserted after an arbitrary city of the sample (in our implementation: the first of the sample). The resulting tour does not respect the constraint that all the segments of a layer must be printed before printing the next layer. This explains the counter performance of our method for this type of instances.

Notice that good solutions are not difficult to obtain. A POPMUSIC method optimising a few layers at a time works well, as soon as the initial solution is already partially sorted, with all the points of a layer being placed before the points of the next layer. For instance, the results presented in Tables 2 and 3 were obtained with a relatively limited computational effort by exploiting the fact that the head has to go to a cleaning point every two layers. So, it is possible to solve a few hundred of small, independent TSP instances instead of a single, large one.

5. Conclusion

This work proposes a method for generating moderately good TSP solutions in $n\ log\ n$. The method does not require assumptions about the problem structure. For the first time, to our knowledge, instances with more than 2 billion of cities have been tackled with a metaheuristic. The method can be used for generating good candidate edges for advanced local searches. It is shown on classical TSP and CTSP instances in the literature that a high proportion of the edges of the best solutions known can be obtained

with a few dozens of runs of our method. The LKH TSP Solver release 2.0.9 includes the proposed method for generating candidate edges sets.

Since the method evaluates the distance between a limited set of city pairs, it is evident that pathological instances exist for this method. The article introduces a new way to model a manufacturing process under a standard TSP. It is analysed why large TSP instances derived from this additive manufacturing process are ill-conditioned. Experiments with 3D printing instances containing few layers show that important production gains are still possible, compared to methods recently proposed.

Future works should study how to solve more efficiently and faster these TSP instances. Indeed, additive manufacturing is increasingly used, but the process is quite slow. Printing a single part can take several hours. The proportion of the time spent by the printing head in non-productive moves can be significant.

Acknowledgement

This research was partially supported by the Swiss National Science Foundation, project 200021_169085. The author thanks anonymous reviewers who help to improve the quality of the paper.

References

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (1999). *Concorde: A code for solving Traveling Salesman Problems.* Retrieved from `http://www.math.princeton.edu/tsp/concorde.html`

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

Applegate, D. L., Cook, W., & Rohe, A. (2003). Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing*, *15*(1), 82-92. doi: 10.1287/ijoc.15.1.82.15157

Arora, S. (1998, September). Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and Other Geometric Problems. *J. ACM*, *45*(5), 753–782. doi: 10.1145/290179.290180

Cacchiani, V., Contreras-Bolton, C., & Toth, P. (2020). Models and Algorithms for the Traveling Salesman Problem with Time-dependent Service times. *European Journal of Operational Research*, *283*(3), 825-843. doi: https://doi.org/10.1016/j.ejor.2019.11.046

Campbell, J. F., Corberán, Á., Plana, I., Sanchis, J. M., & Segura, P. (2021). Solving the Length constrained K-Drones Rural Postman Problem. *European Journal of Operational Research*, *292*(1), 60-72. doi: https://doi.org/10.1016/j.ejor.2020.10.035

Cook, W. J. (2012). *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation.* Princeton University Press.

Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2020). A Hybrid Genetic Algorithm for the Traveling Salesman Problem with Drone. *Journal of Heuristics*, *26*(2), 219–247. doi: 10.1007/s10732-019-09431-y

Helsgaun, K. (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, *1*.

Helsgaun, K. (2014). *Best known solutions to DIMACS TSP instances.* Retrieved from `http://www.akira.ruc.dk/~keld/research/LKH/DIMACS_results.html` (Last updated: October 6, 2014)

Helsgaun, K. (2016). *Helsgaun's implementation of Lin-Kernighan.* Retrieved from `http://webhotel4.ruc.dk/~keld/research/LKH/` (Version LKH-2.0.7)

Helsgaun, K. (2018). *Using POPMUSIC for Candidate Set Generation in the Lin-Kernighan-Helsgaun TSP Solver* (Tech. Rep.). Department of Computer Science, Roskilde University, DK-4000 Roskilde, Denmark.

Johnson, D. S., & McGeoch, L. A. (1997). The Traveling Salesman Problem: A Case Study. In E. Aarts & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 215–310). Wiley.

Johnson, D. S., McGeoch, L. A., & Rothberg, E. E. (1996). Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman bound. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 341–350).

Laporte, G., & Palekar, U. (2002). Some Applications of the Clustered Travelling Salesman Problem. *The Journal of the Operational Research Society*, *53*(9), 972–976.

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, *21*(2), 498–516.

Merz, P., & Huhse, J. (2008). An Iterated Local Search Approach for Finding Provably Good Solutions for Very Large TSP Instances. In G. Rudolph, T. Jansen, N. Beume, S. Lucas, & C. Poloni (Eds.), *Parallel Problem Solving from Nature – PPSN X* (pp. 929–939). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-87700-4_92

Punnen, A. P., & Gutin, G. (2007). *The Traveling Salesman Problem and Its Variations* (1st ed.). Springer US.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling Salesman Problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, *211*(3), 427 - 441. doi: https://doi.org/10.1016/j.ejor.2010.09.010

Taillard, É. D. (2017). TSP Neighbourhood Reduction with POPMUSIC. In *Metaheuristic International Conference (MIC'17) Proceedings* (pp. 237–240).

Taillard, É. D., & Helsgaun, K. (2019). POPMUSIC for the Travelling Salesman Problem. *EURO Journal of Operational Research*, *272*(2), 420–429. doi: 10.1016/j.ejor.2018.06.039

Voigt (Ed.). (1832). *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.* Ilmenau, Germany: Voigt.

Volpato, N., Galvão, L. C., Nunes, L. F., Souza, R. I., & Oguido, K. (2020). Combining heuristics for tool-path optimisation in material extrusion additive manufacturing. *Journal of the Operational Research Society*, *71*(6), 867–877. doi: 10.1080/01605682.2019.1590135

(Éric D. Taillard) HEIG-VD, Case postale 521, Department of Information Technology, University of Applied Sciences of Western Switzerland, Route de Cheseaux 1, CH-1401 Yverdon, Switzerland.