Improved Filtering for the Euclidean Traveling Salesperson Problem in CLP(FD)

Alessandro Bertagnon, Marco Gavanelli
Department of Engineering, University of Ferrara
Via Saragat 1, 44122 Ferrara, Italy
{alessandro.bertagnon, marco.gavanelli}@unife.it

Abstract

The Traveling Salesperson Problem (TSP) is one of the best-known problems in computer science. The Euclidean TSP is a special case in which each node is identified by its coordinates on the plane and the Euclidean distance is used as cost function.

Many works in the Constraint Programming (CP) literature addressed the TSP, and use as benchmark Euclidean instances; however the usual approach is to build a distance matrix from the points coordinates, and then address the problem as a TSP, disregarding the information carried by the points coordinates for constraint propagation.

In this work, we propose to use geometric information, present in Euclidean TSP instances, to improve the filtering power. In order to have a declarative approach, we implemented the filtering algorithms in Constraint Logic Programming on Finite Domains (CLP(FD)).

1 Introduction

The Traveling Salesperson Problem (TSP) is one of the best known problems in computer science: given a graph with a set of nodes, the objective is to find a path that visits all the nodes exactly once and minimizes the traveled distance. Currently, the best solver for the TSP is the mathematical programming solver Concorde (Applegate et al. 2001), exploiting several techniques amongst which Integer Linear Programming and Local Search.

The TSP is a Constraint Optimization Problem (COP); an important technique developed in Artificial Intelligence to solve such problems is constraint propagation (Mackworth 1977). The idea was so successful that new languages were embedding it as part of their operational semantics; for example the Constraint Logic Programming (CLP) (Jaffar and Maher 1994) class of languages was defined to declaratively address constraint problems, and many languages of this class exploited constraint propagation to solve efficiently constraint problems. The CLP research area was later extended to include also imperative and object-oriented programming languages, generating the Constraint Programming (CP) research area.

The TSP was also addressed in the CP literature (Caseau and Laburthe 1997; Kaya and Hooker 2006; Benchimol et al. 2012; Fages and Lorca 2012; Fages, Lorca, and Rousseau 2016; Deudon et al. 2018), but Concorde is still the state of the art, in particular for large instances. Nevertheless, also efficiently solving small or medium size instances is important in various applications (Caseau and Laburthe 1997).

CP approaches are also more flexible: while Concorde can address only *pure TSPs*, i.e., no further side constraints are allowed, in CP many variants can be easily cast, such as the TSP with Time Windows (TSPTW) (Caseau and Koppstein 1993; Pesant et al. 1998; Focacci, Lodi, and Milano 2002b; 2002a), in which cities must be visited within given temporal intervals. Finally, improving the performance of CP models is an important target *per se*.

Some significant sub-classes of the general TSP are the *Metric TSP*, in which the distance function between cities satisfies the triangle inequality, and the *Euclidean TSP*, in which the nodes of the graph represent points in the plane and the distance function is the Euclidean distance. These are reasonable assumptions in many cases, and various instances in the TSPLIB (Reinelt 1991) fall into these classes. From the computational complexity point of view, both the Metric and the Euclidean TSP are NP-hard (Garey, Graham, and Johnson 1976); nonetheless, differently from the general TSP, the Euclidean TSP admits a Polynomial Time Approximation Scheme (PTAS) (Arora 1996; Mitchell 1999). Notwithstanding these theoretically important results, solvers that address the general TSP (e.g., Concorde) are faster in practice.

In CP, no specialized pruning algorithms tailored to the Euclidean TSP have been proposed, and the usual way to tackle Euclidean TSPs is to compute the distance matrix and address the problem as a general TSP. It is worth noting that in the Euclidean TSP more information is available than in the general TSP: the coordinates of the points to be visited are known, and geometrical concepts (straight line segments, angles, etc.) can be defined in the Euclidean plane. Such knowledge has been exploited to improve the search effectiveness: Deudon et al. (2018) train a Deep Neural Network with points coordinates to learn efficient heuristics to explore the search space.

This paper, instead, is the first attempt (to the best of our knowledge) to exploit geometric information to obtain further pruning in CP during the solution of Euclidean TSPs. We address the pure Euclidean TSP (with no side constraints), and propose new constraints that reduce the search space. We also show that the pruning we introduce is not subsumed by that of important works in the area (Benchimol et al. 2012). A further added value is that all constraints and algorithms presented here were implemented in a declarative language, namely CLP on Finite Domains (CLP(FD)).

The rest of the paper is organized as follows. After some preliminaries, we discuss related works (Section 3), we propose new constraints to avoid crossings (Section 4) and to exploit convex hull reasoning (Section 5). We show experimental results in Section 6 and, finally, we conclude.

2 Preliminaries and notation

Let $G = (V, E, w)$ be a weighted graph, where V is a set of nodes, E is a set of edges, and $w : E \mapsto \mathbb{R}^+$. A *path* in G is a sequence $p_{v_{s_0}\text{-}v_{s_k}} = v_{s_0} e_{s_0,s_1} v_{s_1} \dots e_{s_{k-1},s_k} v_{s_k}$ such that (*i*) $v_{s_0}, v_{s_1}, \dots, v_{s_k} \in V$ and are all distinct, and (*ii*) $e_{s_0,s_1}, \dots, e_{s_{k-1},s_k} \in E$. Since a path is uniquely identified by the sequence of its nodes (or of its edges) in the proper order, to simplify the notation we will often write paths as sequences of nodes. The length of a path p is the sum of the weights of its edges: $L(p) = \sum_{i=0}^{k-1} w(e_{s_i,s_{i+1}})$. Given a path $p_{v_{s_0}\text{-}v_{s_k}}$, the sequence obtained by appending e_{s_k,s_0} to a path $p_{v_{s_0}\text{-}v_{s_k}}$ is also called a *circuit c*.

In the Euclidean case, let $\mathcal{P} = \{P_1, \dots, P_n\}$ be a set of points, where $P_i = (x_i, y_i)$. The graph associated with $\mathcal{P}$ is $G^{\mathcal{P}} = (\mathcal{P}, E^{\mathcal{P}}, w^{\mathcal{P}})$, where $E^{\mathcal{P}} = \{e_{i,j} \equiv (P_i, P_j) \mid P_i, P_j \in \mathcal{P}, i \neq j\}$ and $w(P_i, P_j) = d(P_i, P_j)$, where d is the Euclidean distance.

We denote with $\overline{P_iP_j}$ the segment in the plane with endpoints P_i and P_j. Since in the Euclidean case each edge of a graph corresponds to a segment in the plane between the corresponding endpoints, we will often confuse the edge $e_{i,j}$ with the corresponding segment $\overline{P_iP_j}$. Also, we will sometimes confuse the index $i \in V$ with the corresponding point P_i in the plane. We denote with $\overleftrightarrow{P_iP_j}$ the (infinite straight) line passing through points P_i and P_j, and with $\angle P_iP_jP_k$ the counterclockwise angle formed by the segments $\overline{P_iP_j}$ and $\overline{P_jP_k}$ with vertex in P_j from P_i to P_k.

The *convex hull* of a set of points in a Euclidean space is the minimum convex set containing all the points. In the plane it corresponds to a convex polygon and it is completely defined by its vertices. Given a set of points P, we denote with $\mathcal{H}(P) = \langle H_0, H_1, \dots, H_{|\mathcal{H}(P)|-1} \rangle$ the sequence of vertexes of the convex hull in clockwise order.

3 Related work

In the CP literature, three main representations have been devised for defining variables in the Hamiltonian circuit problem and the TSP: the *permutation* representation, the *successor* representation and the *set variable* representation (Benchimol et al. 2012). The last was also extended to the graph representation (Dooms, Deville, and Dupont 2005; Fages and Lorca 2012; Fages, Lorca, and Rousseau 2016), in which also the set of nodes has a lower and an upper bound; however in the TSP case the set of nodes is fixed. In this paper, we use the successor representation, but we will test other representations in future work.

In the successor representation, n variables $Next_i$ are defined; variable $Next_i$ represents the node that follows node i in the circuit, and its initial domain is $\{1, \dots, n\} \setminus \{i\}$. The constraint model includes an `alldifferent`($Next$) constraint (Régin 1994) on the $Next$ array of variables, that ensures that each node has exactly one incoming edge, as well as a `circuit`($Next$) (Beldiceanu and Contejean 1994; Caseau and Laburthe 1997; Kaya and Hooker 2006) constraint (sometimes called `nocycle`) that avoids sub-tours, i.e., cycles of length less than n. In some cases, the constraint model includes, as redundant representation, also a set of $Prev$ variables: $Prev_i$ represents the node that precedes node i in the circuit.

Usually, CP formulations are not as effective in exploiting the objective function as Integer Programming models. Various works use relaxations of the TSP to prune suboptimal branches; the classical relaxations of the TSP are the assignment problem and the one-tree relaxation. Caseau and Laburthe (1997) propose a simple and effective rule for the `circuit` constraint, and also filter values using the assignment-based and the spanning tree relaxation. Kaya and Hooker (2006) propose a new filtering rule, based on graph separators, for the `circuit` constraint. Francis and Stuckey (2014) consider various propagation algorithms for `circuit` and provide for each explanations in the context of a lazy clause generation solver.

Pesant et al. (1998) address the TSPTW and exploit the `circuit` constraint together with the minimum spanning tree relaxation. Focacci, Lodi, and Milano (2002a; 2002b) propose reduced costs filtering to optimization constraints, and in particular use the assignment problem and the minimum spanning forest relaxation.

The groundbreaking work in this area is that by Benchimol et al. (2012): it is the first in which a CP model was able to solve large TSP instances. They propose an implementation of the Weighted Circuit Constraint (WCC) that includes the Held and Karp (1970) scheme, iterates a Lagrangian relaxation to obtain a high-quality one-tree, and uses it to remove edges similarly to reduced costs filtering. It also identifies as mandatory those edges that, if removed, would increase the current lower bound over the quality of the incumbent solution. To find quickly a solution, they first run the Lin-Kernighan-Helsgaun (LKH) algorithm (Lin and Kernighan 1973; Helsgaun 2000). Fages, Lorca, and Rousseau (2016) further improve by casting the problem in CP(Graph) and by means of improved search heuristics.

Fages and Lorca (2012) model Asymmetric TSPs by means of a reduced graph, whose nodes are the Strongly Connected Components of the original graph. In the reduced graph, transitive arcs can be safely removed, and the Minimum Spanning Tree relaxation can be made tighter.

After the paper submission deadline, another paper was published (Isoart and Régin 2019) introducing a new structural constraint in the WCC based on k-cutsets.

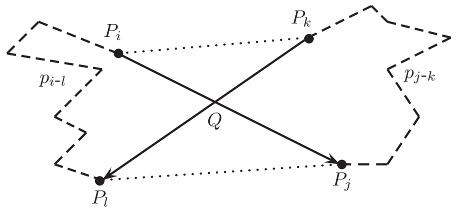

Figure 1: A self-crossing circuit.

4 Avoiding crossings

A well known result in the literature (Flood 1956; Arora 1996) is that the optimal solution of a Metric TSP in the plane cannot include two edges that cross each other.

Theorem 1. *(Flood 1956). Let c^* be an optimal tour of a Metric TSP. Then, for each $e_{i,j}, e_{k,l} \in c^*$ such that $\{i, j, k, l\}$ are all different, the segments $\overline{P_iP_j} \cap \overline{P_kP_l} = \emptyset$.*

Proof. (sketch). In Fig. 1, instead of taking $\overline{P_iP_j}$ and $\overline{P_kP_l}$, a shorter tour chooses the dotted edges $\overline{P_iP_k}$ and $\overline{P_lP_j}$. $\square$

The nocrossing constraint

Theorem 1 suggests to avoid, during the search for an optimal TSP, those paths that include crossing edges. We propose the `nocrossing` constraint, that imposes that a pair of segments in the TSP should not cross each other (except, possibly, in one endpoint). In the successor representation, it is defined as follows:

$$\texttt{nocrossing}(i, Next_i, j, Next_j) = \left(\overline{P_iP_{Next_i}} \cap \overline{P_jP_{Next_j}}\right) \subset \{P_i, P_j\}. \quad (1)$$

A possible propagator for the `nocrossing` constraint would be activated each time a value is removed from the domain of $Next_i$ and would remove inconsistent values from the domain of $Next_j$. Such propagator would be naive in terms of time complexity: propagating it would have the usual cost of arc-consistency propagation for a single constraint of $O(d^2)$ (if d is the size of the domains) in each activation of the constraint.

However, from the definition of arc-consistency and Eq. 1, a value v can be removed from the current domain $Dom(Next_i)$ only if $\overline{P_iP_v}$ intersects all possible segments originating from P_j. A necessary condition for this is that all segments originating from P_j lie on the same half-plane with respect to the line $\overleftrightarrow{P_iP_j}$.

Theorem 2. *Let $o, u \in Dom(Next_i)$ such that P_o and P_u do not lie on the line l passing through P_i and P_j and are in different half-planes with respect to the line l.*

Then, any value $k \in Dom(Next_j)$ such that $P_k \notin l$ is arc-consistent with respect to the constraint $\texttt{nocrossing}(i, Next_i, j, k)$.

Proof. By contradiction, suppose there exists $k \in Dom(Next_j)$ that is inconsistent with the constraint $\texttt{nocrossing}(i, Next_i, j, k)$. Since it is inconsistent, the segment $\overline{P_jP_k}$ must cross all the segments $\overline{P_iP_z}$ such that $z \in Dom(Next_i)$, so in particular it crosses both $\overline{P_iP_o}$ and $\overline{P_iP_u}$. But the intersection $I_o \equiv \overline{P_iP_o} \cap \overline{P_jP_k}$ lies on a different half-plane from the one that hosts $I_u \equiv \overline{P_iP_u} \cap \overline{P_jP_k}$, so P_k lies in both half-planes, meaning that $P_k \in l$: absurd. $\square$

Theorem 2 does not cover the case in which one of the points P_o, P_u lies on the line $\overleftrightarrow{P_iP_j}$. The following proposition takes care of the cases in which three points are aligned.

Proposition 1. *Given a graph G whose nodes do not lie all on the same line; let a, b and c be three nodes of G such that $P_c \in \overline{P_aP_b}$. Then, segment $\overline{P_aP_b}$ is not in the optimal TSP.*

Proof. Omitted, based on the triangle inequality. $\square$

Given Proposition 1, it is possible to remove from the domain of each variable $Next_a$ all the values b such that the segment $\overline{P_aP_b}$ contains another node of the graph G. In the rest of the paper, we will assume that such pre-processing step has been done before the search starts; this assumption simplifies the following exposition.

Algorithm 1 sketches the algorithm of a propagator for the `nocrossing` constraint; it is awakened when the domain of variable $Next_i$ is reduced, and it performs propagation to possibly reduce the domain of $Next_j$; to fully implement the constraint, another symmetric propagator would be imposed on the reverse direction (from $Next_j$ to $Next_i$). In the first phase the propagator is suspended waiting that all elements in the domain of $Next_j$ lie on the same half-plane with respect to $\overline{P_iP_j}$. To do so, one element $Over \in Dom(j)$ and one $Under \in Dom(j)$ that lie, respectively, over and under the line $\overleftrightarrow{P_iP_j}$ are selected. If one of them does not exist, all possible segments originating from P_j lie on the same half-plane and the control passes to the next phase; otherwise, the propagator suspends waiting that one of the two elements $Over$ and $Under$ is removed from $Dom(Next_j)$. This strategy mimics, in a sense, the idea of watched literals proposed in SAT solvers (Moskewicz et al. 2001) (and currently used also in other solvers).

In case all elements in the domain of $Next_i$ lie on the same half-plane with respect to the line $\overleftrightarrow{P_iP_j}$, the algorithm steps to phase 2 (line 9). Also in phase 2, there is a necessary condition for arc-consistency of the `nocrossing` propagator to remove values from the domain of $Next_j$; it is based on comparing angles with vertex in P_j and subtended by segments originating from P_i:

Theorem 3. *For each $k \in Dom(Next_i) \cup Dom(Next_j)$, let $\alpha_k = \angle P_iP_jP_k$. If all the elements $q \in Dom(Next_i)$ lie on the same half-plane with respect to the line $\overleftrightarrow{P_iP_j}$, then a necessary condition for a segment $\overline{P_jP_t}$ originating from P_j and reaching an element $t \in Dom(Next_j)$ to cross all segments $\overline{P_iP_q}$ originating from P_i is that $\alpha_t \leq \underline{\alpha}$, where $\underline{\alpha} = \min\{\alpha_q \mid q \in Dom(Next_i)\}$.*

Proof. Consider a coordinate system centered into P_j, with the abscissa pointing toward P_i and such that all the points in $Dom(Next_i)$ have non-negative ordinate (see Figure 2).

Algorithm 1 nocrossing_propagator

```
1: function NOCROSSING_PROP($i, Next_i, j, Next_j$)
2:     $Under \leftarrow$ select one element in $Dom(Next_i)$
                 s.t. $P_{Under}$ is under the line $\overleftrightarrow{P_iP_j}$
3:     if there is no such element then
4:         phase2($i, Next_i, j, Next_j$)
5:     else $Over \leftarrow$ select one element in $Dom(Next_i)$
                 s.t. $P_{Over}$ is over the line $\overleftrightarrow{P_iP_j}$
6:         if there is no such element then
7:             phase2($i, Next_i, j, Next_j$)
8:         else suspend waiting for either $Over$ or $Under$
                 to be removed from $Dom(Next_i)$
9: function PHASE2($i, Next_i, j, Next_j$)
10:    $\underline{\alpha} \leftarrow \min\{\alpha_x \mid x \in Dom(Next_i)\}$
11:    $x^i_{\underline{\alpha}} \leftarrow$ value in $Dom(Next_i)$ corresponding to $\underline{\alpha}$
12:    $\overline{\beta} \leftarrow \max\{\beta_x \mid x \in Dom(Next_i)\}$
13:    $x^i_{\overline{\beta}} \leftarrow$ value in $Dom(Next_i)$ corresponding to $\overline{\beta}$
14:    for all $y^j \in Dom(Next_j)$ s.t. $\alpha_{y^j} < \underline{\alpha}$ do
15:        if $\beta_{y^j} > \overline{\beta}$ then
16:            remove $y^j$ from $Dom(Next_j)$
17:    if $|Dom(Next_i)| > 1 \wedge |Dom(Next_j)| > 1$ then
18:        suspend waiting for either $x^i_{\underline{\alpha}}$ or $x^i_{\overline{\beta}}$ to be removed
               from $Dom(Next_i)$
```

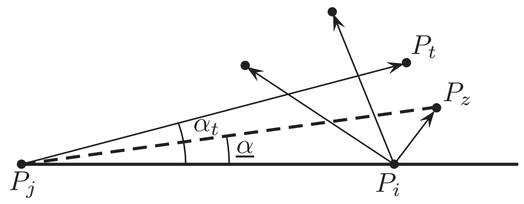

Figure 2: Thm 3. An arrow from P_x to P_y means that $y \in Dom(Next_x)$. Dashed lines are plotted to show the angles.

By contradiction, suppose that $\alpha_t > \underline{\alpha}$; we prove that there is a segment originating from P_i that does not intersect with $\overline{P_jP_t}$. Let $z \in Dom(Next_i)$ such that $\underline{\alpha} = \angle P_iP_jP_z$.

In polar coordinates, the segment $\overline{P_iP_z}$ is seen from P_j with angles between 0 and $\underline{\alpha}$. All the points on the segment $\overline{P_jP_t}$ are seen under the angle α_t. Since $\alpha_t > \underline{\alpha}$, there is no intersection between $\overline{P_iP_z}$ and $\overline{P_jP_t}$. □

Clearly, the condition of Theorem 3 is not sufficient, as shown in Figure 3. A sufficient condition is Theorem 4.

Theorem 4. *For each $k \in Dom(Next_i) \cup Dom(Next_j)$, let $\beta_k = \angle P_kP_iP_j$. Assume all the elements $q \in Dom(Next_i)$ lie on the same half-plane with respect to the line $\overleftrightarrow{P_iP_j}$. Suppose there exists $t \in Dom(Next_j)$ such that $\alpha_t < \underline{\alpha}$.*

Then a sufficient condition for segment $\overline{P_jP_t}$ to cross all segments $\overline{P_iP_q}$ such that $q \in Dom(Next_i)$ is that $\beta_t > \overline{\beta}$, where $\overline{\beta} = \max\{\beta_q \mid q \in Dom(Next_i)\}$.

Proof. By contradiction, suppose $\exists k \in Dom(Next_i)$ such

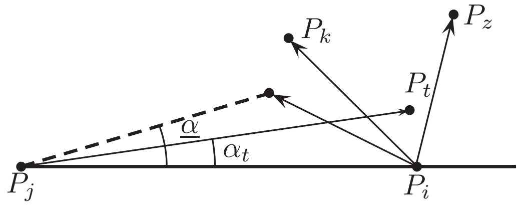

Figure 3: The condition in Thm 3 is not sufficient: $\alpha_t < \underline{\alpha}$ but $\overline{P_jP_t}$ does not cross all segments exiting from P_i.

that $\overline{P_iP_k}$ does not intersect $\overline{P_jP_t}$ (Figure 4). Since $\alpha_t < \underline{\alpha} \leq \alpha_k$, P_k and P_i lie on different sides of the ray $\overrightarrow{P_jP_t}$.

In order not to have crossing between $\overline{P_iP_k}$ and $\overline{P_jP_t}$, both P_j and P_t must lie on the same half-plane with respect to $\overrightarrow{P_iP_k}$. Since $\beta_j = 0$, and all points in the domain of $Next_i$ are above the x axis, $\beta_j < \beta_k$, thus to be on the same half-plane, also $\beta_t < \beta_k$. But $\beta_t > \overline{\beta} \geq \beta_k$: contradiction. □

Thanks to Theorems 3 and 4, we can obtain a better complexity with respect to the naive algorithm. Note that all angles can be pre-computed before starting the search, thus avoiding to compute trigonometric functions during search. It is also possible to pre-compute and store in a linked list the elements in the (initial) domains of the two variables sorted with respect to their α angle. In this way the computation of the minimum in line 10 of Algorithm 1 amounts to finding the first element in the linked list that belongs to $Dom(Next_i)$; assuming that domain membership is checked in constant time, the search for the minimum in line 10 can be done in $O(d)$ amortized time on one branch of the search tree. The same can be done with a linked list containing the elements in $Dom(Next_i)$ sorted with respect to their β angle: also line 12 is executed in $O(d)$ amortized time on one branch of the search tree. The loop in lines 14-16 scans the linked list and is stopped as soon as one element is found whose angle is greater than or equal to $\underline{\alpha}$; assuming that the removal of a domain element is done in constant time, and since the comparison in line 15 takes constant time, the whole loop has $O(d)$ complexity per each activation of the propagator of phase 2 (to be compared to the $O(d^2)$ of the naive propagator). Since the propagator is activated when at least one element is removed from $Dom(Next_i)$, such propagator is awakened at most $O(d)$ times in a branch of the search tree, giving phase 2 an overall complexity of $O(d^2)$ amortized time over one branch of the search tree.

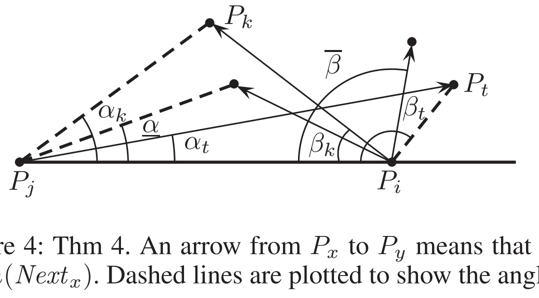

Figure 4: Thm 4. An arrow from P_x to P_y means that $y \in Dom(Next_x)$. Dashed lines are plotted to show the angles.

5 Convex Hull Reasoning

One consequence of Theorem 1 is that if the nodes of the graph do not lie on the same line, nodes on the boundary of the convex hull are always visited in their cyclic order in the optimal TSP (Deineko, van Dal, and Rote 1994).

To simplify the exposition, we will present the following pruning in the context of symmetry breaking constraints, although similar reasoning could be performed also without breaking symmetries.

In the successor representation, the same TSP can be represented by two symmetric solutions that differ only for the order (clockwise or counterclockwise) in which the nodes are visited. One way to break this symmetry is to fix one direction (e.g., clockwise); in such a case, the convex hull reasoning is an effective way to impose the clockwise order.

To compute the convex hull $\mathcal{H}(P)$, we used Andrew (1979) monotone chain, with complexity $O(n \log n)$. We devised three ways to exploit the information about the hull for propagation.

The simplest is to impose that the successor of a convex hull vertex cannot be another member of $\mathcal{H}(P)$ except for the one that immediately follows it: $\forall i \in [0, |\mathcal{H}(P)| - 1]$, $Dom(Next_{H_i}) \cap \mathcal{H}(P) \subseteq \{H_{(i+1) \bmod |\mathcal{H}(P)|}\}$. These constraints are unary, so they are equivalent to a domain reduction, without any overhead during search.

The second is reasoning on the angle formed by the incoming and the outgoing arcs in hull vertexes: in order to visit nodes in a clockwise order, the angle between the incoming edge and the outgoing edge of H_i cannot be positive (it must be between $-\pi$ and 0) or, stated otherwise, it must correspond to a right turn. In case the constraint model includes also the $Prev$ variables, one can observe that the angle formed by the outgoing edge and any reference direction must be less than the angle formed by the incoming edge with the same reference direction. This produces a propagator in the same spirit of the classical *less-than* propagator: simply compute the minimum angle in $Dom(Next_h)$ and remove from $Dom(Prev_h)$ the elements associated with a smaller (or equal) angle (Algorithm 2). A symmetric propagator takes care of the opposite direction (from $Prev_h$ to $Next_h$). Again, note that all angles are pre-computed before search, and the search for the minimum takes $O(d)$ amortized time over one branch of the search tree.

If the constraint model does not include the $Prev$ variables, then one can wait for $Next_h$ to become ground; when the arc outgoing from P_h is fixed, the value h is removed from the domain of all other variables $Next_i$ such that the angle $\angle P_i P_h P_{Next_h}$ would correspond to a left turn.

The third is obtained by imposing that each vertex in a path originating from a point H_i cannot reach any vertex in H except for $H_{(i+1) \bmod |\mathcal{H}(P)|}$. The propagator is imposed for each pair $(H_i, H_{(i+1) \bmod |\mathcal{H}(P)|})$. The implementation of this propagator is inspired by the `circuit` constraint (Caseau and Laburthe 1997), but performs more powerful pruning thanks to the convex hull reasoning. If a partial path has been defined starting from H_i up to a node j, and such path does not contain vertex $H_{(i+1) \bmod |\mathcal{H}(P)|}$, then the variable $Next_j$ cannot take any value in $\mathcal{H}(P)$ except for $H_{(i+1) \bmod |\mathcal{H}(P)|}$ (Algorithm 3). If the partial path reaches the next vertex of the convex hull, the constraint is entailed and exits the constraint store (line 3). In the propagator for the `circuit` constraint (Caseau and Laburthe 1997), instead, from the domain of the variable $Next_j$ only the initial value (in our case, H_i) of the path is removed.

Algorithm 2 clockwise_angle_propagator

```
Require: $P_h$ to be in the perimeter of the convex hull
 1: function CLOCKWISE_ANGLE_PROP($h$, $Next$, $Prev$)
 2:     $m \leftarrow \min\{\alpha_v \mid v \in Dom(Next_h)\}$
 3:     $Dom(Prev_h) \leftarrow Dom(Prev_h) \setminus \{i \mid \alpha_i \leq m\}$
 4:     if $Next_h$ and $Prev_h$ are not ground then
 5:         suspend until $m$ is removed from $Dom(Next_h)$
```

Algorithm 3 hull_path_propagator

```
Require: $P_{End}$ to be in the perimeter of the convex hull.
 1: function HULL_PATH_PROP($Start, End, \mathcal{H}(P), Next$)
 2:     if $Start$ == $End$ then
 3:         return $true$
 4:     if $Next_{Start}$ is ground then
 5:         hull_path_prop($Next_{Start}, End, \mathcal{H}(P), Next$)
 6:     else
 7:         remove $(\mathcal{H}(P) \setminus \{End\})$ from $Dom(Next_{Start})$
 8:     suspend waiting for $Next_{Start}$ to become ground
```

Extensions of the convex-hull reasoning

Now that we have propagators that exploit the knowledge about the convex hull, we wish to extend their applicability also to points that lie in the interior of the hull.

A consequence of the absence of crossings is that the optimal TSP is a simple polygon, and it divides the plane into exactly two areas: an *internal* and an *external* area. Imagine now to cut the optimal TSP with two vertical lines (parallel to the y axis): the stripe between the two lines will contain alternate internal and external regions. The borders (i.e., the parts of the circuit inside the stripe) will be alternatively visited left-to-right (or clockwise) and right-to-left (or counterclockwise). In order to exploit this informal intuition for pruning, we provide the following theorem:

Theorem 5. *Suppose that (e.g., during search) a partial path $p_{s\text{-}e}$ has been defined, starting in node s and ending in node e. Consider the polygon Q delimited by such path and by the segment $\overline{P_s P_e}$, and suppose that such polygon is a simple polygon, i.e., no two edges intersect. Suppose that the partial path $p_{s\text{-}e}$ touches its vertexes in clockwise order.*

Let $F \subset V$ be the set of nodes whose corresponding points lie in the interior of polygon Q, $I = F \cup \{s, e\}$ and let $\mathcal{H}(I) = \langle H_0^I \equiv e, H_1^I, \ldots, H_{k-1}^I, H_k^i \equiv s \rangle$ be the sequence of vertexes of its convex hull, in counterclockwise order.

Suppose that the convex hull $\mathcal{H}(I)$ does not intersect the path $p_{s\text{-}e}$, except for the endpoints P_s and P_e.

Then any non self-crossing tour containing the path $p_{s\text{-}e}$ reaches the vertexes in $\mathcal{H}(I)$ in the order $H_0^I, \ldots H_k^I$.

Proof. By contradiction, suppose that a non self-crossing circuit $c^* \supseteq p_{s\text{-}e}$ reaches the vertexes of H^I in an order different from their sequence order. W.l.o.g., suppose that after vertex H^I_j, the next vertex of H^I reached by the tour c^* is H^I_{j+2}; i.e., vertex H^I_{j+1} is not reached in the order of H^I. Let $p_{H^I_j\text{-}H^I_{j+2}} \subset c^*$ be the path connecting H^I_j and H^I_{j+2}.

Consider the polygon R (dotted, in Fig. 5) delimited by: (*i*) the path $p_{s\text{-}e}$ (*ii*) the perimeter of the convex hull $\mathcal{H}(I)$ from P_e to H^I_j (*iii*) the path $p_{H^I_j\text{-}H^I_{j+2}}$ (*iv*) the perimeter of the convex hull $\mathcal{H}(I)$ from H^I_{j+2} to P_s.

Clearly, the point H^I_{j+1} lies in the interior of R. In order to reach it without self-crossings, c^* cannot pass through $p_{s\text{-}e}$ nor $p_{H^I_j\text{-}H^I_{j+2}}$. On the other hand, c^* cannot cross the boundary of the $\mathcal{H}(I)$ between P_e and H^I_j (and from H^I_{j+2} to P_s) because, by definition of convex hull, there are no vertexes to be visited that are interior to polygon Q and that do not belong to $\mathcal{H}(I)$. So, c^* cannot reach H^I_{j+1}. □

If we find an *internal* hull $\mathcal{H}(I)$, we can then apply the previous propagators (Algorithms 2 and 3) also to the vertexes of $\mathcal{H}(I)$, with the obvious care that if $p_{s\text{-}e}$ reaches points in clockwise order, then the vertexes of the $\mathcal{H}(I)$ will be reached in counterclockwise order (and vice-versa). We maintain all partial paths during the search, and we compute a convex hull for each of these paths. In this way, we are orthogonal to heuristics (we can use any search heuristic without invalidating our propagation).

In the implementation, we applied the pruning on internal hulls only when the polygon Q is convex (differently from the Q polygon depicted in Figure 5). Checking the convexity amounts to check that each turn in the path is on the same side (a right turn, in case of a clockwise path), simplifies finding if a point is inside the polygon, and also lets us avoid the check that $\mathcal{H}(I)$ does not intersect $p_{s\text{-}e}$. In future work, we will experiment also with non-convex polygons.

The time complexity of our implementation of the extended convex hull is $O(n^2)$ to compute the points inside the polygon, then we use Andrew (1979) monotone chain ($O(n \log n)$) to find the hull. We currently recompute $\mathcal{H}(I)$ from scratch after each decision.

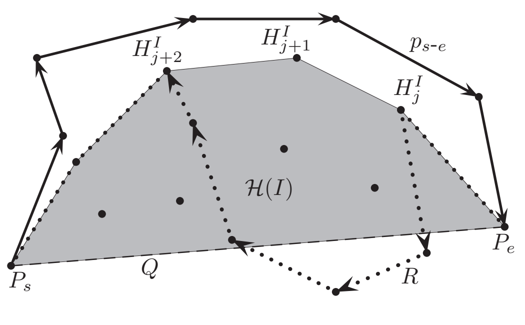

Figure 5: From Theorem 5: the path $p_{s\text{-}e}$ is the current assignment. Polygon Q is delimited by $p_{s\text{-}e}$ and the (dashed) segment $\overline{P_e P_s}$. I contains the points inside Q plus P_s and P_e; $\mathcal{H}(I)$ (gray in the picture) is its convex hull. Polygon R is delimited by $p_{s\text{-}e}$ and the dotted segments.

6 Experimental evaluation

To assess the effectiveness of the proposed algorithms, we devised experiments based on randomly-generated TSPs and structured instances.

All algorithms are implemented in the ECL^{i}PS^e CLP language (Schimpf and Shen 2012). All constraint models are based on the successor representation (with the `circuit` constraint and `alldifferent` (Puget 1998) for improved pruning) with both *Next* and *Prev* variables, that are linked through the `inverse` constraint.

The constraint model named `Geometric` includes also the `nocrossing` constraint, the removal of aligned points according to proposition 1, and the `clockwise` constraint that implements the propagation described in Section 5, that also acts as symmetry breaking constraint in this model.

We compare our model with a simple model (named `CLP(FD)` in the following) that (besides the `alldifferent`, `circuit` and `inverse` constraint) includes as symmetry breaking the constraint $Next_1 < Prev_1$ used by Benchimol et al. (2012).

In order to show that the pruning we provide is not subsumed by that of state of the art techniques, we implemented in ECL^{i}PS^e, in the successor representation, also the Held and Karp bound with pruning based on reduced and marginal costs, as proposed by Benchimol et al. (2012) (shown with `BvHRRR` in the following).

Concerning search strategies, we use the *max-regret* (Caseau and Laburthe 1997) and the state-of-the-art `LC_FIRST MAX_COST` (Fages, Lorca, and Rousseau 2016), based on *Last Conflict* (Lecoutre et al. 2009). As (Benchimol et al. 2012), we also experimented injecting the upper bound given by the LKH (v. 2.0.7) algorithm.

All tests were run on ECL^{i}PS^e v. 7.0, build #48, with a time limit of 1800s on Intel® Xeon® E5-2630 v3 CPUs running at 2.4GHz, using only one core and with 1GB of reserved memory.

Structured instances

We consider all the instances taken from the TSPLIB, the Concorde website[1] and the CITIES dataset[2] up to 100 nodes. These sources provide various types of instances, amongst which Euclidean ones, represented as sets of points in the plane, and geographical ones, represented as sets of points (with latitude and longitude) on the surface of the Earth. We selected all the Euclidean instances and also added those geographical instances in which the cities to be visited lie on a limited part of the geoid, so that the geographical distance can be approximated with the Euclidean distance.

Table 1 reports the results; we omitted those instances in which no algorithm was able to reach the optimal solution within the time limit of 1800s. Except for easy instances, the constraint models containing geometric filtering are the ones that solve the instances to optimality in the shortest time. These results show the positive interaction between the geometric filtering and that performed by `BvHRRR` alone.

[1]http://www.math.uwaterloo.ca/tsp/world/countries.html

[2]https://people.sc.fsu.edu/~jburkardt/datasets/cities/cities.html

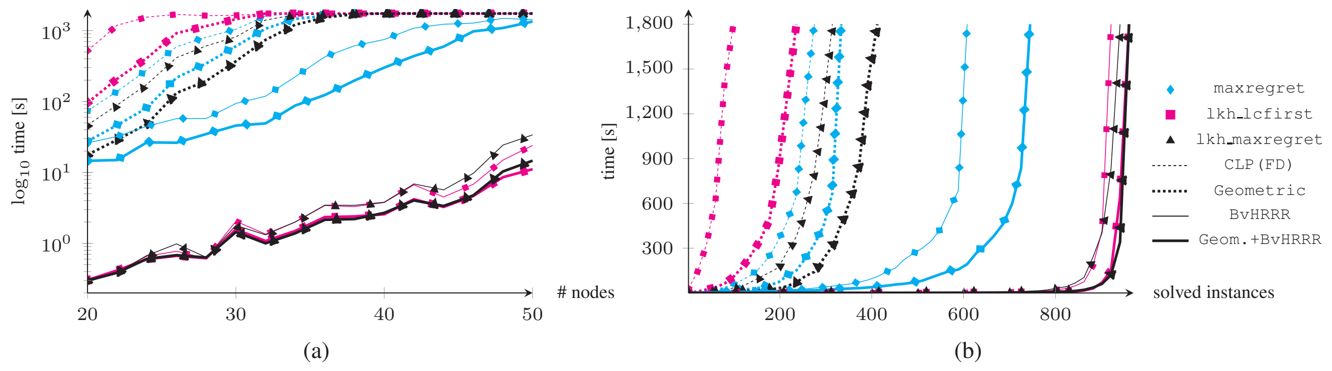

Figure 6: Experimental results on randomly-generated Euclidean TSP instances. (a) Average solving time of filtering algorithms varying the size of the instances and the search strategies. (b) Number of solved instances varying the solving time. In both graphs, the mark distinguishes the various search strategies, solid (opposed to dotted) lines represent usage (resp. not usage) of the `BvHRRR` pruning, while thick (resp. thin) lines represent usage (or not usage) of `Geometric` filtering.

Analyzing instances that were not solved to optimality, we found that our additional pruning is most effective during the proof of optimality, rather than on finding good solutions.

Table 1: Comparing filtering algorithms on structured instances with time limit 1800s. For each instance we report total solving time and number of explored nodes to reach the optimal solution and prove its optimality.

	lkh_lcfirst				lkh_maxregret			
	BvHRRR		Geo+BvH		BvHRRR		Geo+BvH	
instance	time	nodes	time	nodes	time	nodes	time	nodes
uk12	**0.04**	12	0.05	12	**0.04**	**0**	**0.04**	**0**
burma14	0.06	14	0.07	14	**0.05**	**0**	0.07	**0**
ulysses16	**0.07**	16	0.09	16	**0.07**	**0**	0.08	**0**
ulysses22	**0.12**	22	0.15	22	**0.12**	**0**	0.16	**0**
wg22	0.14	22	0.18	22	**0.13**	**0**	0.17	**0**
bayg29	0.45	30	0.44	35	**0.41**	3	0.43	**2**
wi29	**0.23**	29	0.32	29	**0.23**	**0**	0.32	**0**
dj38	**0.42**	38	0.65	38	**0.42**	**0**	0.65	**0**
dantzig42	2.16	48	**1.60**	43	4.83	28	2.11	**3**
att48	3.21	66	**2.38**	55	8.02	36	3.10	**10**
uscap50	**0.64**	50	1.22	50	**0.64**	**0**	1.22	**0**
eil51	T/O	-	523.92	1384	313.53	916	**84.43**	**195**
berlin52	0.86	52	1.67	52	**0.84**	**0**	1.52	**0**
kn57	8.53	180	5.67	120	8.24	38	**5.14**	**18**
wg59	**1.13**	59	1.92	59	1.35	1	1.80	**0**
st70	1411.63	3934	**327.79**	**985**	T/O	-	T/O	-
eil76	72.56	227	33.20	156	160.09	230	**31.80**	**26**
rat99	T/O	-	T/O	-	T/O	-	**436.97**	**574**
rd100	25.81	117	**21.79**	107	52.99	80	21.94	**22**

Random instances

We used the generator of the DIMACS challenge (Johnson and McGeoch 2007), which provides instances in two classes: uniform and clustered. We randomly generated instances from 20 to 50 nodes in steps of 2, in both classes. For each size and class we generated 30 instances.

Figure 6a shows the geometric mean of the runtime of the four algorithms varying the size of the instance and the search strategy. The addition of the filtering on geometric properties roughly halves the runtime, both with respect to the simple `CLP(FD)` and to the advanced pruning based on the Held and Karp (`BvHRRR`). Cactus plots (Figure 6b) show that, when the LKH bound is used, the two search strategies perform in a quite similar way. The introduction of geometric filtering (represented with thick lines in Figure 6) allowed us to solve almost all the instances within the given timeout.

7 Conclusions

In this paper, we proposed to use the geometric information present in Euclidean TSP instances to provide additional pruning with respect to the techniques already available in CP. This, to the best of our knowledge, is the first attempt to exploit such additional information to prune the search space in constraint programming. Note that all algorithms were implemented for a declarative language, ECL^{i}PS^e, without reverting to external imperative languages.

We showed that the pruning we perform is orthogonal with respect to that obtained by Benchimol et al. (2012) in their seminal work, and that adding reasoning on geometrical properties can further reduce the running time. Despite the results that we have achieved and presented in this paper, our approaches are still not competitive with Concorde.

One limitation in the current work is that it is applicable to the pure Euclidean TSP, but not to all its variants, such as the TSP with time windows (Pesant et al. 1998; Focacci, Lodi, and Milano 2002b; 2002a) or the Vehicle Routing Problem (VRP), since their optimal solutions may contain crossings. However, for some problems, extensions of the proposed techniques could be beneficiary; for example in the optimal solution of a Euclidean VRP, the tour of each vehicle is not self-intersecting (although it can cross the path of other vehicles).

Acknowledgments

We are deeply indebted with Joachim Schimpf for helping us in the implementation of some of the constraints. This work was partially supported by GNCS-INdAM.

References

Andrew, A. 1979. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters* 9(5):216–219.

Applegate, D.; Bixby, R. E.; Chvátal, V.; and Cook, W. J. 2001. TSP cuts which do not conform to the template paradigm. In Jünger, M., and Naddef, D., eds., *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School, Schloß Dagstuhl, Germany, 15-19 May 2000]*, volume 2241 of *Lecture Notes in Computer Science*, 261–304. Springer.

Arora, S. 1996. Polynomial time approximation schemes for euclidean TSP and other geometric problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, 2–11.

Beldiceanu, N., and Contejean, E. 1994. Introducing global constraints in CHIP. *Math. Comput. Model.* 20(12):97–123.

Benchimol, P.; van Hoeve, W. J.; Régin, J.; Rousseau, L.; and Rueher, M. 2012. Improved filtering for weighted circuit constraints. *Constraints* 17(3):205–233.

Caseau, Y., and Koppstein, P. 1993. A rule-based approach to a time-constrained traveling salesman problem. In 2^{nd} *Int. Symp. on AI and Mathematics 1992*. Bellcore Technical Memorandum.

Caseau, Y., and Laburthe, F. 1997. Solving small TSPs with constraints. In Naish, L., ed., *Logic Programming, Proceedings of the Fourteenth International Conference on Logic Programming, Leuven, Belgium, July 8-11, 1997*, 316–330. MIT Press.

Deineko, V. G.; van Dal, R.; and Rote, G. 1994. The convex-hull-and-line traveling salesman problem: A solvable case. *Inf. Process. Lett.* 51(3):141–148.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L. 2018. Learning heuristics for the TSP by policy gradient. In van Hoeve, W. J., ed., *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*, volume 10848 of *Lecture Notes in Computer Science*, 170–181. Springer.

Dooms, G.; Deville, Y.; and Dupont, P. 2005. CP(Graph): Introducing a graph computation domain in constraint programming. In van Beek, P., ed., *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, 211–225. Springer.

Fages, J., and Lorca, X. 2012. Improving the asymmetric TSP by considering graph structure. *CoRR* abs/1206.3437.

Fages, J.; Lorca, X.; and Rousseau, L. 2016. The salesman and the tree: the importance of search in CP. *Constraints* 21(2):145–162.

Flood, M. M. 1956. The traveling-salesman problem. *Operations Research* 4.

Focacci, F.; Lodi, A.; and Milano, M. 2002a. Embedding relaxations in global constraints for solving TSP and TSPTW. *Ann. Math. Artif. Intell.* 34(4):291–311.

Focacci, F.; Lodi, A.; and Milano, M. 2002b. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing* 14(4):403–417.

Francis, K. G., and Stuckey, P. J. 2014. Explaining circuit propagation. *Constraints* 19(1):1–29.

Garey, M. R.; Graham, R. L.; and Johnson, D. S. 1976. Some NP-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, 10–22. New York, NY, USA: ACM.

Held, M., and Karp, R. M. 1970. The traveling-salesman problem and minimum spanning trees. *Operations Research* 18:1138–1162.

Helsgaun, K. 2000. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126(1):106–130.

Isoart, N., and Régin, J.-C. 2019. Integration of structural constraints into tsp models. In *International Conference on Principles and Practice of Constraint Programming*, 284–299. Springer.

Jaffar, J., and Maher, M. J. 1994. Constraint logic programming: A survey. *J. Log. Program.* 19/20:503–581.

Johnson, D. S., and McGeoch, L. A. 2007. Experimental analysis of heuristics for the stsp. In *The traveling salesman problem and its variations*. Springer. 369–443.

Kaya, L. G., and Hooker, J. N. 2006. A filter for the circuit constraint. In Benhamou, F., ed., *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, 706–710. Springer.

Lecoutre, C.; Saïs, L.; Tabary, S.; and Vidal, V. 2009. Reasoning from last conflict(s) in constraint programming. *Artif. Intell.* 173(18):1592–1614.

Lin, S., and Kernighan, B. W. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21(2):498–516.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artif. Intell.* 8(1):99–118.

Mitchell, J. S. B. 1999. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.* 28(4):1298–1309.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, 530–535. New York, NY, USA: ACM.

Pesant, G.; Gendreau, M.; Potvin, J.; and Rousseau, J. 1998. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science* 32(1):12–29.

Puget, J. 1998. A fast algorithm for the bound consistency of alldiff constraints. In Mostow, J., and Rich, C., eds., *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, 359–366. AAAI Press / The MIT Press.

Régin, J. 1994. A filtering algorithm for constraints of difference in CSPs. In Hayes-Roth, B., and Korf, R. E., eds., *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, 362–367. AAAI Press / The MIT Press.

Reinelt, G. 1991. TSPLIB - A traveling salesman problem library. *INFORMS Journal on Computing* 3(4):376–384.

Schimpf, J., and Shen, K. 2012. $\mathrm{Ecl^{i}ps^{e}}$ - from LP to CLP. *TPLP* 12(1-2):127–156.