Geometric reasoning on the Traveling Salesperson Problem: comparing Answer Set Programming and Constraint Logic Programming Approaches

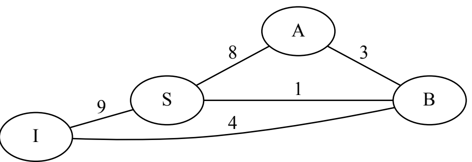
Alessandro Bertagnon Marco Gavanelli
Department of Engineering - University of Ferrara
{alessandro.bertagnon,marco.gavanelli}@unife.it

39TH ITALIAN CONFERENCE ON COMPUTATIONAL LOGIC
JUNE 26-28, 2024 - ROME, ITALY

Travelling Salesperson Problem

Definition (TSP)

Given a graph $G = (N, E)$ with a cost function $f : E \mapsto \mathcal{R}$ find a sequence of nodes such that all nodes are visited, taking only the edges in the graph, and minimizing the cost of the visited edges.



Euclidean Travelling Salesperson Problem

Definition (ETSP)

In a *Euclidean TSP* the nodes are associated to points in the plane. For each pair of nodes there is an edge, and its cost is given by the Euclidean distance.

$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

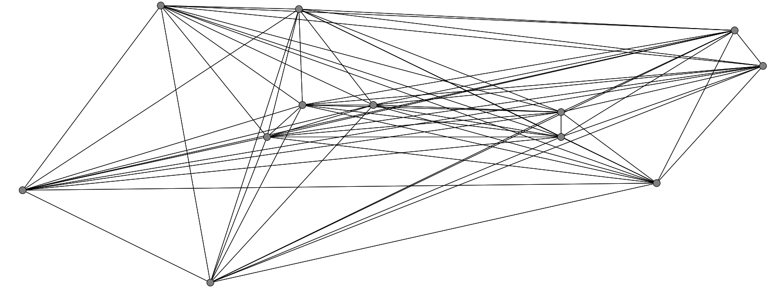
Picture drawn with *ASPECT*! [CILC 2023]
https://github.com/abertagnon/aspect

Euclidean Travelling Salesperson Problem

Definition (ETSP)

In a *Euclidean TSP* the nodes are associated to points in the plane. For each pair of nodes there is an edge, and its cost is given by the Euclidean distance.

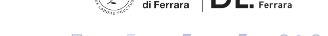
$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$



Picture drawn with *ASPECT*! [CILC 2023]
https://github.com/abertagnon/aspect

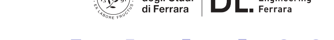
How are ETSPs solved?

- The usual approach
  - compute the distance matrix
  - solve as a normal TSP
- On the other hand, further information is available: coordinates of the nodes
- In the Euclidean plane, concepts from Euclidean geometry can be used: lines, segments, angles, ...
- In [BG20], we exploited such information to provide further pruning in CLP(FD)
- What about ASP?



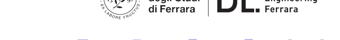
How are ETSPs solved?

- The usual approach
  - compute the distance matrix
  - solve as a normal TSP
- On the other hand, further information is available: coordinates of the nodes
- In the Euclidean plane, concepts from Euclidean geometry can be used: lines, segments, angles, ...
- In [BG20], we exploited such information to provide further pruning in CLP(FD)
- What about ASP?



How are ETSPs solved?

- The usual approach
  - compute the distance matrix
  - solve as a normal TSP
- On the other hand, further information is available: coordinates of the nodes
- In the Euclidean plane, concepts from Euclidean geometry can be used: lines, segments, angles, ...
- In [BG20], we exploited such information to provide further pruning in CLP(FD)
- What about ASP?



How are ETSPs solved?

- The usual approach
    - compute the distance matrix
    - solve as a normal TSP
- On the other hand, further information is available: coordinates of the nodes
- In the Euclidean plane, concepts from Euclidean geometry can be used: lines, segments, angles, ...
- In [BG20], we exploited such information to provide further pruning in CLP(FD)
- What about ASP?

How are ETSPs solved?

- The usual approach
  - compute the distance matrix
  - solve as a normal TSP
- On the other hand, further information is available: coordinates of the nodes
- In the Euclidean plane, concepts from Euclidean geometry can be used: lines, segments, angles, . . .
- In [BG20], we exploited such information to provide further pruning in CLP(FD)
- What about ASP?

ASP approach for the TSP

Input

```
point(P,X,Y).   % (X,Y) are coordinates of point P
cost(P1,P2,C). % there is an edge P1-P2 with cost C
```

TSP

```
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(A,_,_).
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(B,_,_).

reached(1).
reached(B) :- cycle(A,B), reached(B).

:- point(B,_,_), not reached(B).

#minimize{ C,A,B : cycle(A,B), cost(A,B,C) }.
```

ASP approach for the TSP

Input

```
point(P,X,Y).  % (X,Y) are coordinates of point P
cost(P1,P2,C). % there is an edge P1-P2 with cost C
```

TSP

```
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(A,_,_).
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(B,_,_).

reached(1).
reached(B) :- cycle(A,B), reached(B).

:- point(B,_,_), not reached(B).

#minimize{ C,A,B : cycle(A,B), cost(A,B,C) }.
```

ASP approach for the TSP

Input

```
point(P,X,Y).  % (X,Y) are coordinates of point P
cost(P1,P2,C). % there is an edge P1-P2 with cost C
```

TSP

```
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(A,_,_).
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(B,_,_).

reached(1).
reached(B) :- cycle(A,B), reached(B).

:- point(B,_,_), not reached(B).

#minimize{ C,A,B : cycle(A,B), cost(A,B,C) }.
```

ASP approach for the TSP

Input

```
point(P,X,Y).  % (X,Y) are coordinates of point P
cost(P1,P2,C). % there is an edge P1-P2 with cost C
```

TSP

```
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(A,_,_).
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(B,_,_).

reached(1).
reached(B) :- cycle(A,B), reached(B).

:- point(B,_,_), not reached(B).

#minimize{ C,A,B : cycle(A,B), cost(A,B,C) }.
```

ASP approach for the TSP

Input

```
point(P,X,Y).  % (X,Y) are coordinates of point P
cost(P1,P2,C). % there is an edge P1-P2 with cost C
```

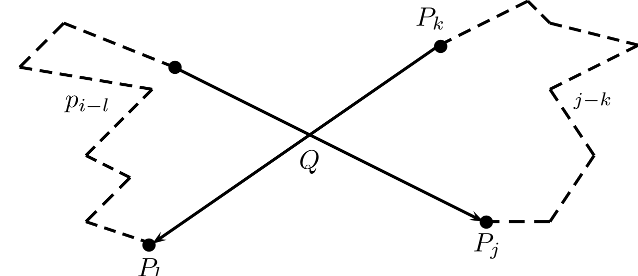
TSP

```
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(A,_,_).
1 { cycle(A,B) : cost(A,B,_) } 1 :- point(B,_,_).

reached(1).
reached(B) :- cycle(A,B), reached(B).

:- point(B,_,_), not reached(B).

#minimize{ C,A,B : cycle(A,B), cost(A,B,C) }.
```

Crossings

Theorem

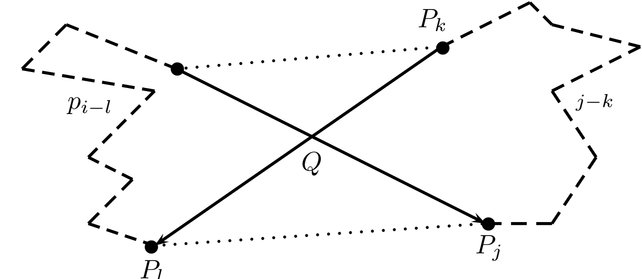
(Flood [Flo56]). The optimal solution of a Euclidean TSP cannot include two edges that cross each other.



Crossings

Theorem

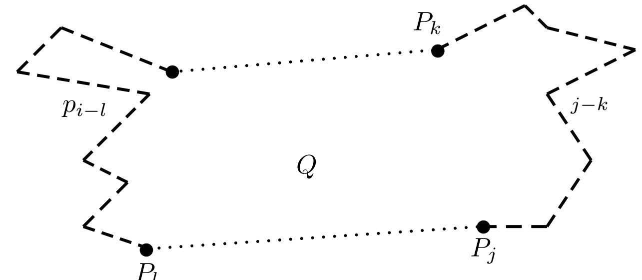
(Flood [Flo56]). The optimal solution of a Euclidean TSP cannot include two edges that cross each other.



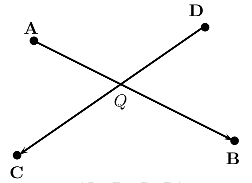
Crossings

Theorem

(Flood [Flo56]). The optimal solution of a Euclidean TSP cannot include two edges that cross each other.



Finding intersections



$$\begin{cases} \mathbf{Q} = \mathbf{A} + r(\mathbf{B} - \mathbf{A}) \\ \mathbf{Q} = \mathbf{C} + s(\mathbf{D} - \mathbf{C}) \end{cases}$$

```
cross(A,B,C,D):-
    point(A,Ax,Ay), point(B,Bx,By), point(C,Cx,Cy), point(D,Dx,Dy),
    Den = (Bx-Ax)*(Dy-Cy)-(By-Ay)*(Dx-Cx),
    NumR = (Ay-Cy)*(Dx-Cx)-(Ax-Cx)*(Dy-Cy),
    NumS = (Ay-Cy)*(Bx-Ax)-(Ax-Cx)*(By-Ay),
    Den != 0,
    NumR * Den > 0,
    NumS * Den > 0,
    |NumR| < |Den|,
    |NumS| < |Den|.
```

Compiled to a set of facts by `gringo`!

Finding intersections

$$\begin{cases} \mathbf{Q} = \mathbf{A} + r(\mathbf{B} - \mathbf{A}) \\ \mathbf{Q} = \mathbf{C} + s(\mathbf{D} - \mathbf{C}) \end{cases}$$

```
cross(A,B,C,D):-
   point(A,Ax,Ay), point(B,Bx,By), point(C,Cx,Cy), point(D,Dx,Dy),
   Den = (Bx-Ax)*(Dy-Cy)-(By-Ay)*(Dx-Cx),
   NumR = (Ay-Cy)*(Dx-Cx)-(Ax-Cx)*(Dy-Cy),
   NumS = (Ay-Cy)*(Bx-Ax)-(Ax-Cx)*(By-Ay),
   Den != 0,
   NumR * Den > 0,
   NumS * Den > 0,
   |NumR| < |Den|,
   |NumS| < |Den|.
```

Compiled to a set of facts by `gringo`!

Finding intersections

A, B, C, D, Q

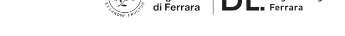
$$\begin{cases} \mathbf{Q} = \mathbf{A} + r(\mathbf{B} - \mathbf{A}) \\ \mathbf{Q} = \mathbf{C} + s(\mathbf{D} - \mathbf{C}) \end{cases}$$

```
cross(A,B,C,D):-
   point(A,Ax,Ay), point(B,Bx,By), point(C,Cx,Cy), point(D,Dx,Dy),
   Den = (Bx-Ax)*(Dy-Cy)-(By-Ay)*(Dx-Cx),
   NumR = (Ay-Cy)*(Dx-Cx)-(Ax-Cx)*(Dy-Cy),
   NumS = (Ay-Cy)*(Bx-Ax)-(Ax-Cx)*(By-Ay),
   Den != 0,
   NumR * Den > 0,
   NumS * Den > 0,
   |NumR| < |Den|,
   |NumS| < |Den|.
```

Compiled to a set of facts by `gringo`!

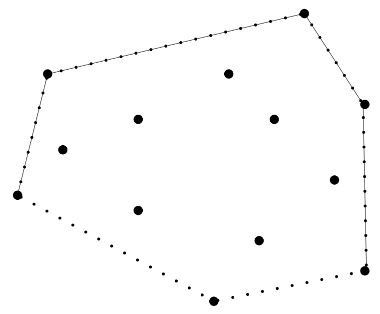
Avoiding crossings

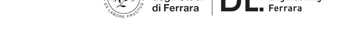
```
:- cycle(A,B), cycle(C,D), cross(A,B,C,D).
```



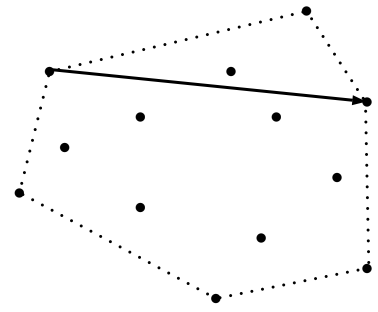
Convex Hull Reasoning

Convex Hull Reasoning

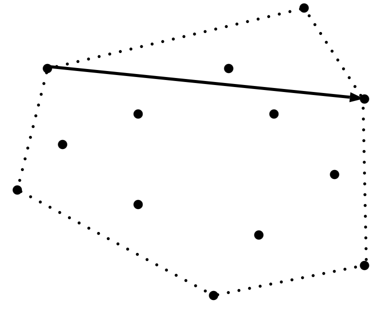




Convex Hull Reasoning



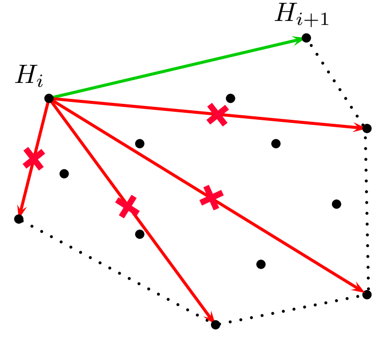
Convex Hull Reasoning



Corollary

(Deineko, van Dal, and Rote [DvDR94]). (Assuming that not all nodes of the graph lie on the same line) Nodes on the boundary of the convex hull are always visited in their cyclic order in an optimal TSP.

Convex Hull Reasoning



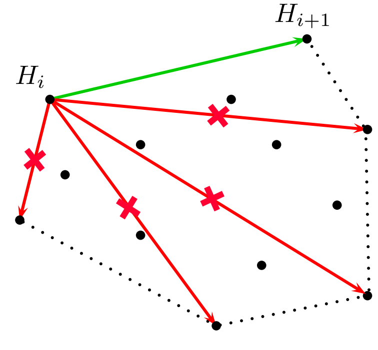
Rule 1

Sort the points on the border of the hull in clockwise order: $\mathcal{H} = [H_1, \ldots H_k]$
In the optimal TSP, the successor of node H_i cannot be any node in $\mathcal{H}$ except H_{i+1} (modulo k)

Code

```
:- not convex_hull_next(A,B), cycle(A,B),
hull(A), hull(B).
```

Convex Hull Reasoning



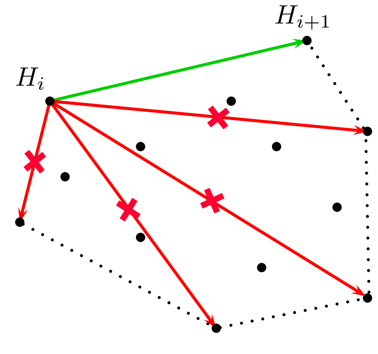
Rule 1

Sort the points on the border of the hull in clockwise order: $\mathcal{H} = [H_1, \ldots H_k]$
In the optimal TSP, the successor of node H_i cannot be any node in $\mathcal{H}$ except H_{i+1} (modulo k)

Code

```
:- not convex_hull_next(A,B), cycle(A,B),
hull(A), hull(B).
```

Convex Hull Reasoning



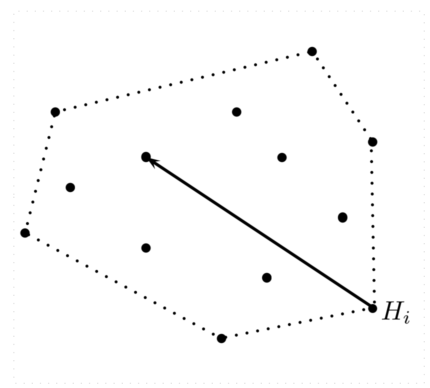
Rule 1

Sort the points on the border of the hull in clockwise order: $\mathcal{H} = [H_1, \ldots H_k]$
In the optimal TSP, the successor of node H_i cannot be any node in $\mathcal{H}$ except H_{i+1} (modulo k)

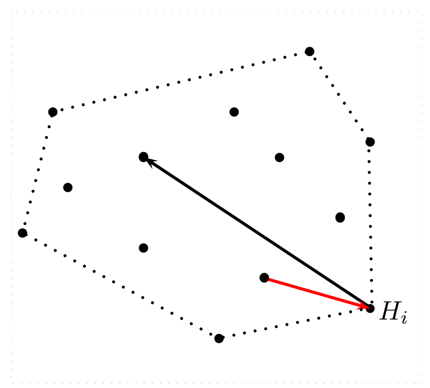
Code

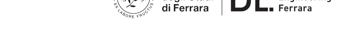
```
:- not convex_hull_next(A,B), cycle(A,B),
hull(A), hull(B).
```

Angles on the Hull

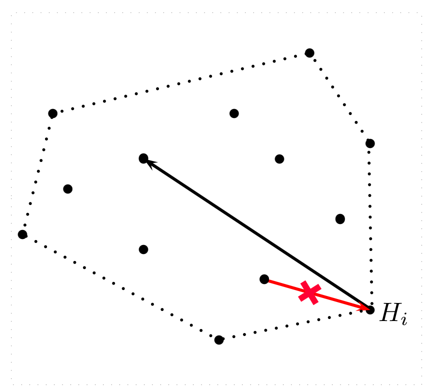


Angles on the Hull

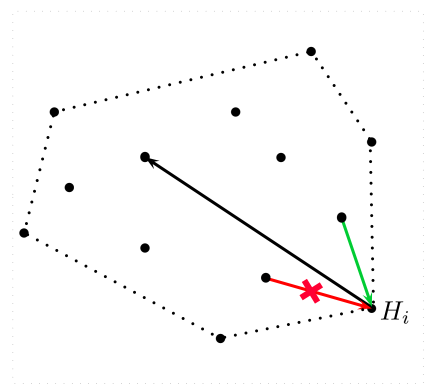




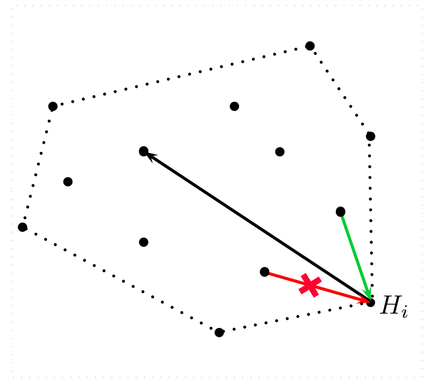
Angles on the Hull



Angles on the Hull



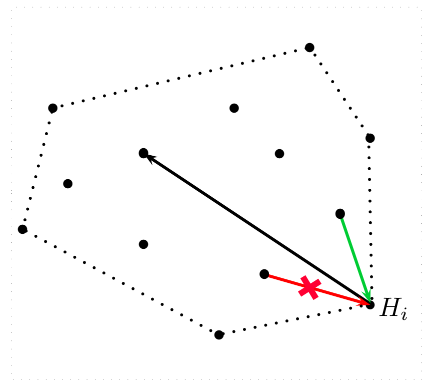
Angles on the Hull



Code

```
:- hull(X), cycle(From,X),
cycle(X,To), left_turn(From,X,To).

left_turn(From,P,To):- point(From,X,Y),
    point(P,X0,Y0), point(To,X1,Y1),
    (Y-Y0)*(X1-X0) - (X-X0)*(Y1-Y0)>0.
```

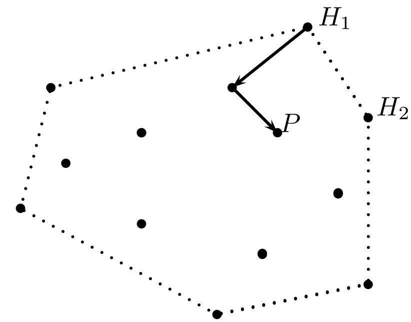
Angles on the Hull

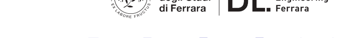


Code

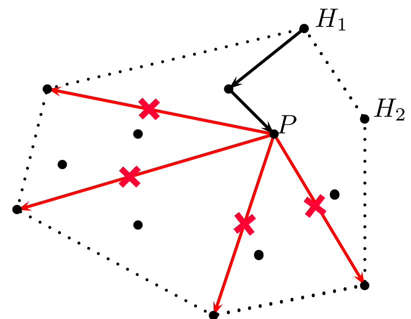
```
:- hull(X), cycle(From,X),
cycle(X,To), left_turn(From,X,To).

left_turn(From,P,To):- point(From,X,Y),
    point(P,X0,Y0), point(To,X1,Y1),
    (Y-Y0)*(X1-X0) - (X-X0)*(Y1-Y0)>0.
```

Paths to the Hull

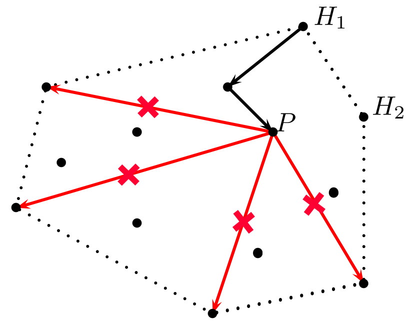




Paths to the Hull



Paths to the Hull

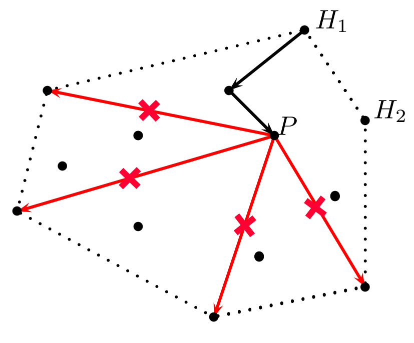


Code

```
path_hull(H1,P):- hull(H1),
    cycle(H1,P), not hull(P).
path_hull(H1,P):- hull(H1),
    path_hull(H1,B), cycle(B,P),
    not hull(P).
```

Università degli Studi di Ferrara | DE Department of Engineering Ferrara

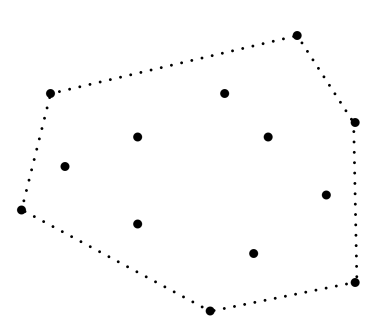
Paths to the Hull



Code

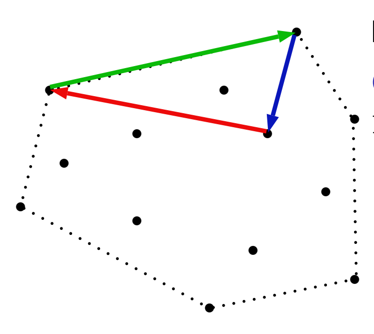
```
path_hull(H1,P):- hull(H1),
    cycle(H1,P), not hull(P).
path_hull(H1,P):- hull(H1),
    path_hull(H1,B), cycle(B,P),
    not hull(P).

:- path_hull(H1,P),
   cycle(B,C), hull(C),
   not convex_hull_next(H1,C).
```

Computing the Hull



The Hull can be pre-computed externally, or it can be computed in ASP

Computing the Hull

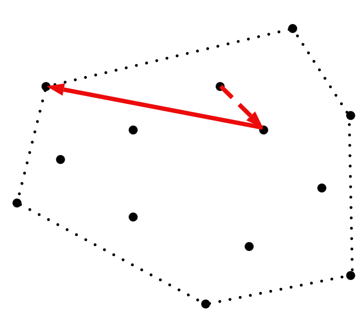


The Hull can be pre-computed externally, or it can be computed in ASP

Code

```
hull(P) :- point(P,_,_), not inside(P).
```

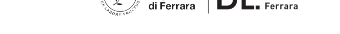
Computing the Hull



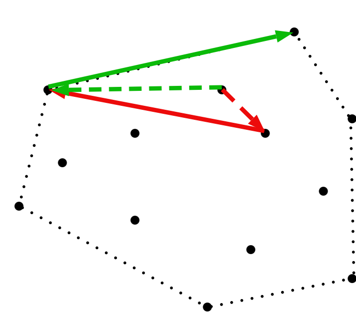
The Hull can be pre-computed externally, or it can be computed in ASP

Code

```
hull(P) :- point(P,_,_), not inside(P).
inside(P):-point(P1,_,_),turn_right(P,P1,P2)
    point(P2,_,_), turn_right(P,P2,P3),
    point(P3,_,_), turn_right(P,P3,P1).
```



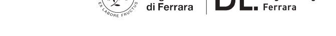
Computing the Hull



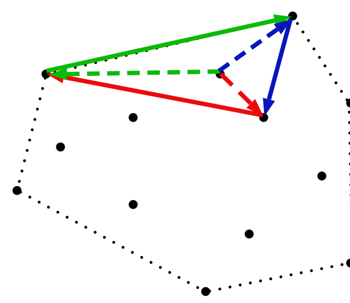
The Hull can be pre-computed externally, or it can be computed in ASP

Code

```
hull(P) :- point(P,_,_), not inside(P).
inside(P):-point(P1,_,_),turn_right(P,P1,P2)
     point(P2,_,_), turn_right(P,P2,P3),
     point(P3,_,_), turn_right(P,P3,P1).
```



Computing the Hull

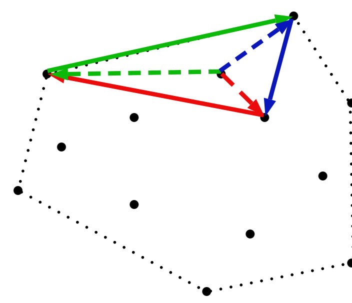


The Hull can be pre-computed externally, or it can be computed in ASP

Code

```
hull(P) :- point(P,_,_), not inside(P).
inside(P):-point(P1,_,_),turn_right(P,P1,P2)
    point(P2,_,_), turn_right(P,P2,P3),
    point(P3,_,_), turn_right(P,P3,P1).
```

Computing the Hull

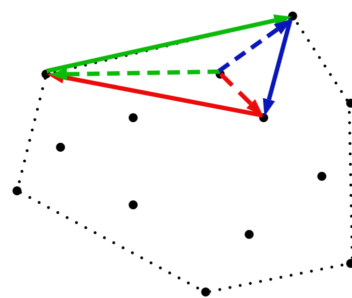


The Hull can be pre-computed externally, or it can be computed in ASP

Code

```
hull(P) :- point(P,_,_), not inside(P).
inside(P):-point(P1,_,_),turn_right(P,P1,P2)
    point(P2,_,_), turn_right(P,P2,P3),
    point(P3,_,_), turn_right(P,P3,P1).

convex_hull_next(A,B):- hull(A), hull(B),
    A != B, not turn_right(A,B,_).
```

Computing the Hull



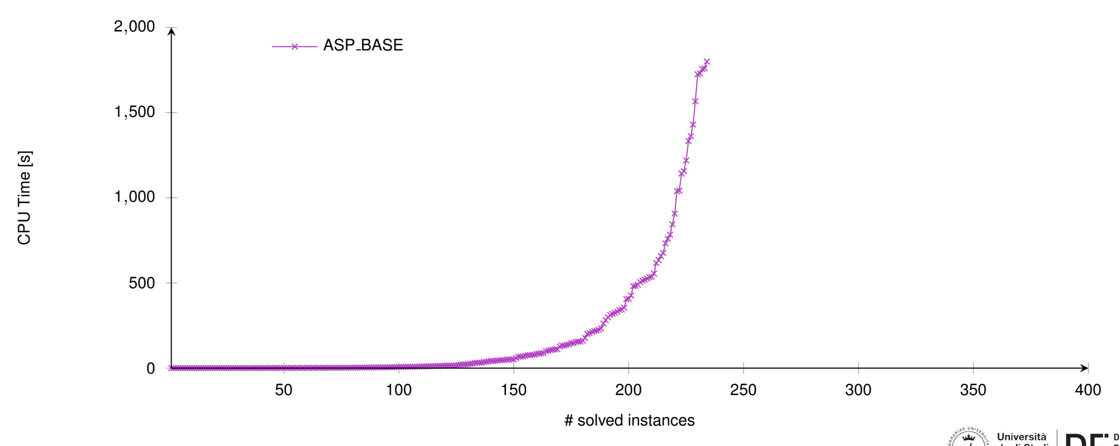
The Hull can be pre-computed externally, or it can be computed in ASP

Code

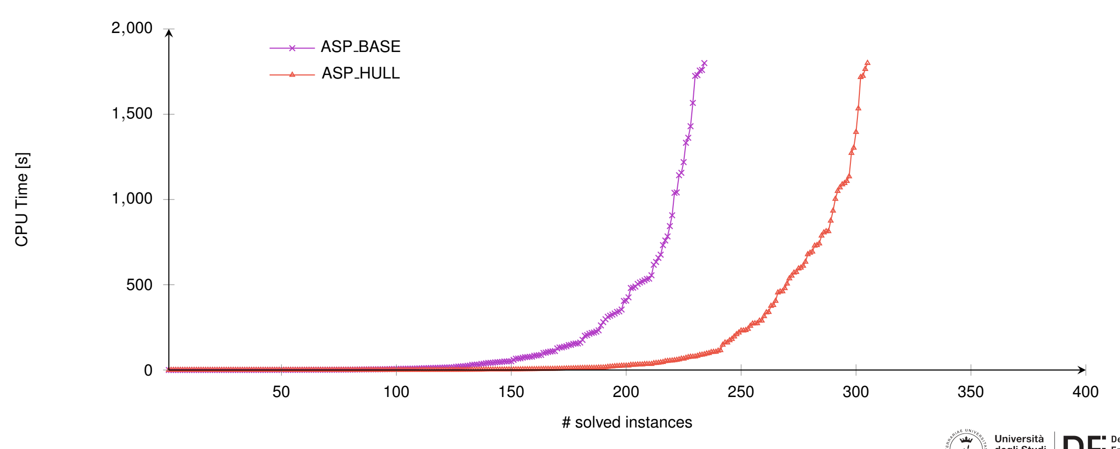
```
hull(P) :- point(P,_,_), not inside(P).
inside(P):-point(P1,_,_),turn_right(P,P1,P2)
    point(P2,_,_), turn_right(P,P2,P3),
    point(P3,_,_), turn_right(P,P3,P1).

convex_hull_next(A,B):- hull(A), hull(B),
    A != B, not turn_right(A,B,_).
```

Convex hull computed by gringo!

Experiments

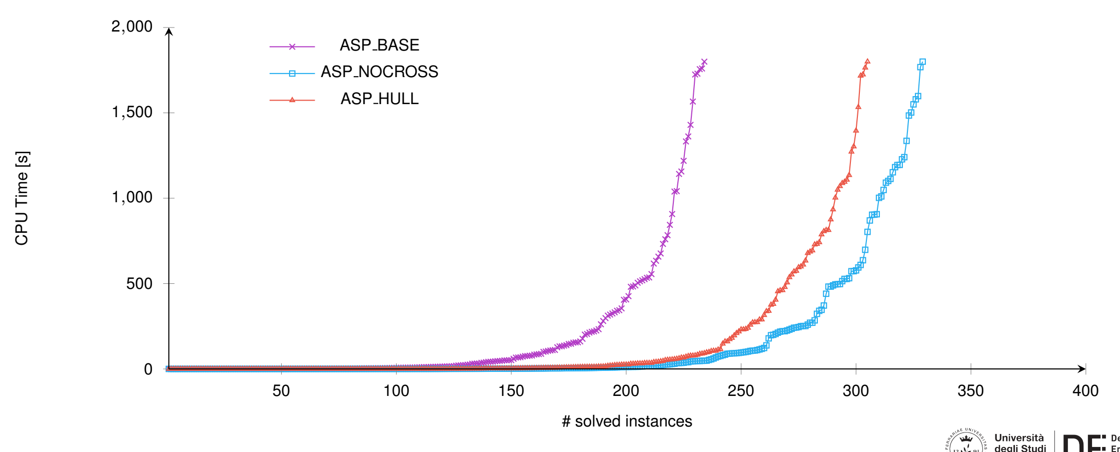


Experiments



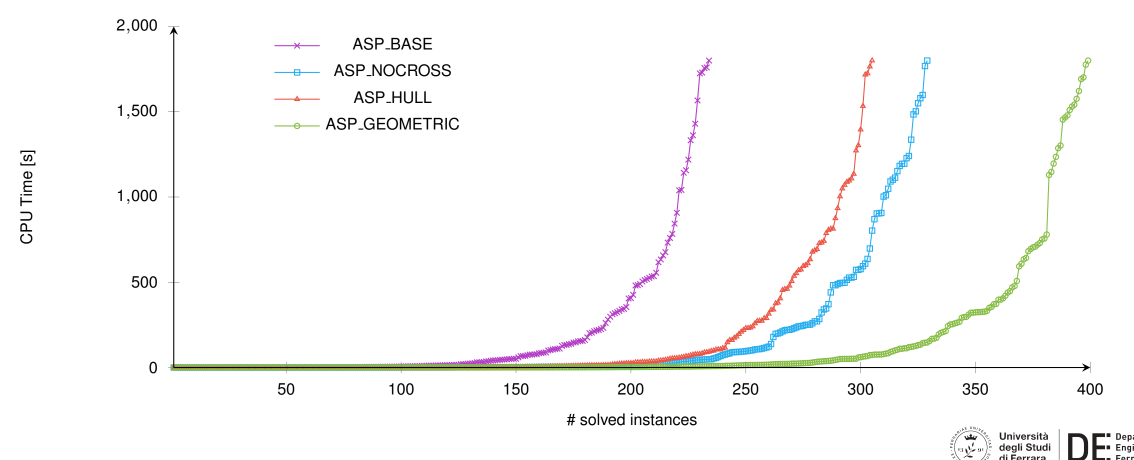
Università degli Studi di Ferrara
DE Department of Engineering Ferrara

Experiments



Università degli Studi di Ferrara
DE Department of Engineering Ferrara

Experiments



Grounding time

CPU Time [s]

10^2

10^0

10^{-2}

ASP_BASE grounding

ASP_BASE solving

Instances (sorted in ascending order by size)

Grounding time

CPU Time [s]

10^2

10^0

10^{-2}

ASP_BASE grounding

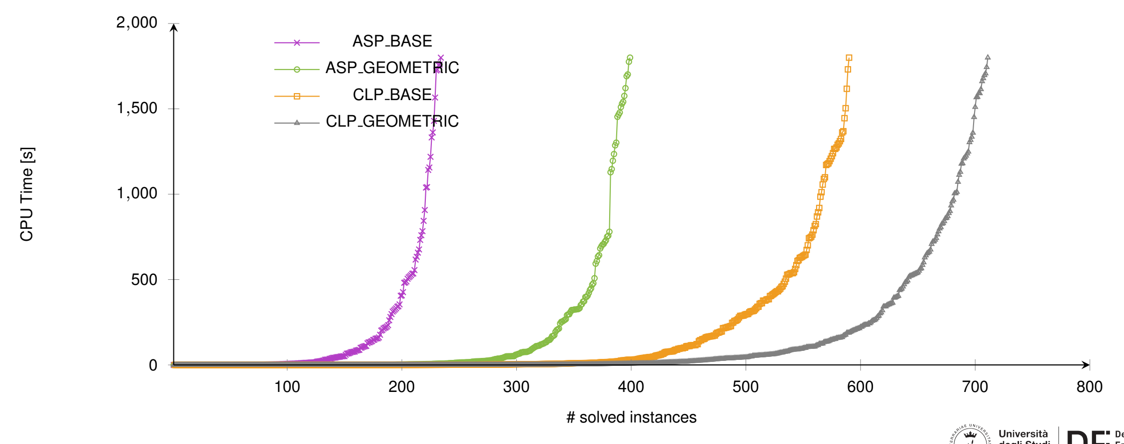
ASP_GEOMETRIC grounding

ASP_BASE solving

ASP_GEOMETRIC solving

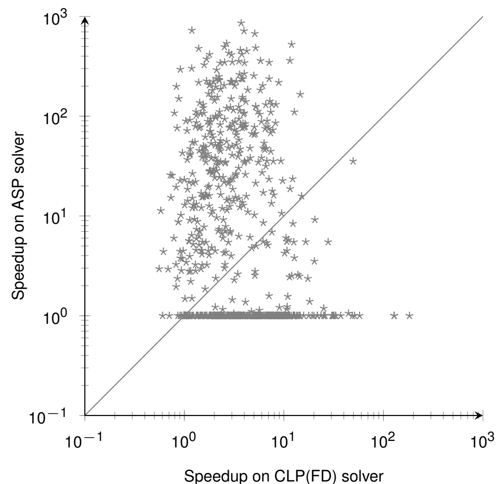
Instances (sorted in ascending order by size)

comparison ASP and CLP(FD)



Università degli Studi di Ferrara
DE Department of Engineering Ferrara

Speedup in ASP and CLP(FD)



Conclusions

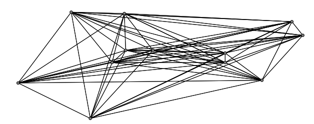
- Applied geometric reasoning to Euclidean TSP solving. Speedups from 1 to 3 orders of magnitude were obtained w.r.t. the basic ASP encoding
- The CLP(FD) approach [BG20] still faster, but was not trivial (based on theorems reported in the paper), took several development time, about 1500 lines of code.
- The ASP version was very quick to develop, and about 30 lines of ASP code

Future work

- Study other Euclidean routing problems (Vehicle routing? Generalized-TSP?)
- use machine learning to add only the most useful nocrossing constraints pair
- Extend the notions to geographic instances

THANK YOU FOR YOUR ATTENTION!

Marco



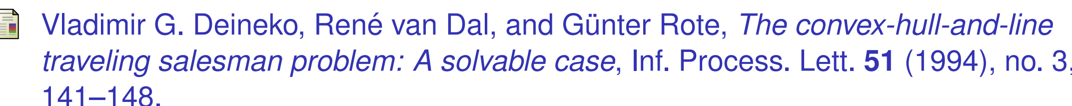
https://github.com/abertagnon/aspect

References I

Alessandro Bertagnon and Marco Gavanelli, *Improved filtering for the euclidean traveling salesperson problem in CLP(FD)*, The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 1412–1419.

Jill Cirasella, David S. Johnson, Lyle A. McGeoch, and Weixiong Zhang, *The asymmetric traveling salesman problem: Algorithms, instance generators, and tests*, Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Washington, DC, USA, January 5-6, 2001, Revised Papers (Adam L. Buchsbaum and Jack Snoeyink, eds.), Lecture Notes in Computer Science, vol. 2153, Springer, 2001, pp. 32–59.

References II



- Vladimir G. Deineko, René van Dal, and Günter Rote, *The convex-hull-and-line traveling salesman problem: A solvable case*, Inf. Process. Lett. **51** (1994), no. 3, 141–148.
- M. M. Flood, *The traveling-salesman problem*, Operations Research **4** (1956).

Parameters of experiments

To generate realistic Euclidean TSP instances, we used the generator of the DIMACS challenge [CJMZ01], that provides instances in two classes: *uniform* and *clustered*. We randomly generated instances from 10 to 35 nodes, in both classes. For each size and class, we generated 16 instances.

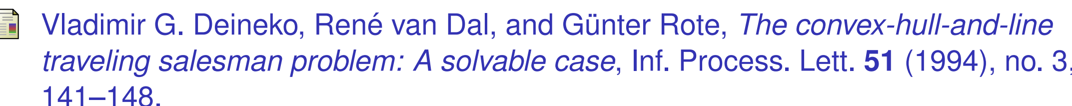
References I

Alessandro Bertagnon and Marco Gavanelli, *Improved filtering for the euclidean traveling salesperson problem in CLP(FD)*, The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 1412–1419.

Jill Cirasella, David S. Johnson, Lyle A. McGeoch, and Weixiong Zhang, *The asymmetric traveling salesman problem: Algorithms, instance generators, and tests*, Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, Washington, DC, USA, January 5-6, 2001, Revised Papers (Adam L. Buchsbaum and Jack Snoeyink, eds.), Lecture Notes in Computer Science, vol. 2153, Springer, 2001, pp. 32–59.

Università degli Studi di Ferrara | DE: Department of Engineering Ferrara

References II



Vladimir G. Deineko, René van Dal, and Günter Rote, *The convex-hull-and-line traveling salesman problem: A solvable case*, Inf. Process. Lett. **51** (1994), no. 3, 141–148.

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