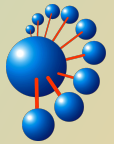


Improving the state-of-the-art in the Traveling Salesman Problem: An Anytime Automatic Algorithm Selection

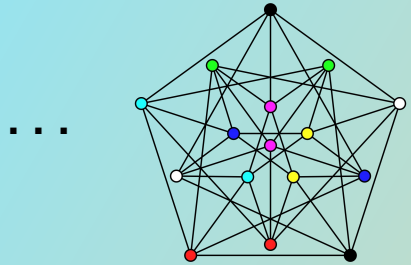
Isaías Huerta, Roberto Asín-Achá,
Julio Godoy, Daniel Neira, Daniel Ortega, Vicente Varas

Expert Systems with Applications

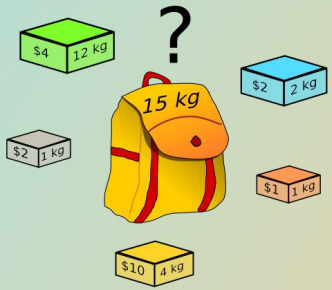
Universidad de Concepción | August, 26

Automatic Algorithm Selection

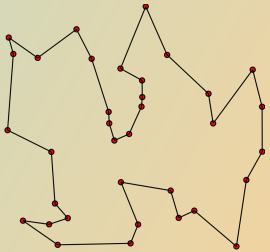
Combinatorial Optimization Problems



Graph Coloring



Knapsack



TSP

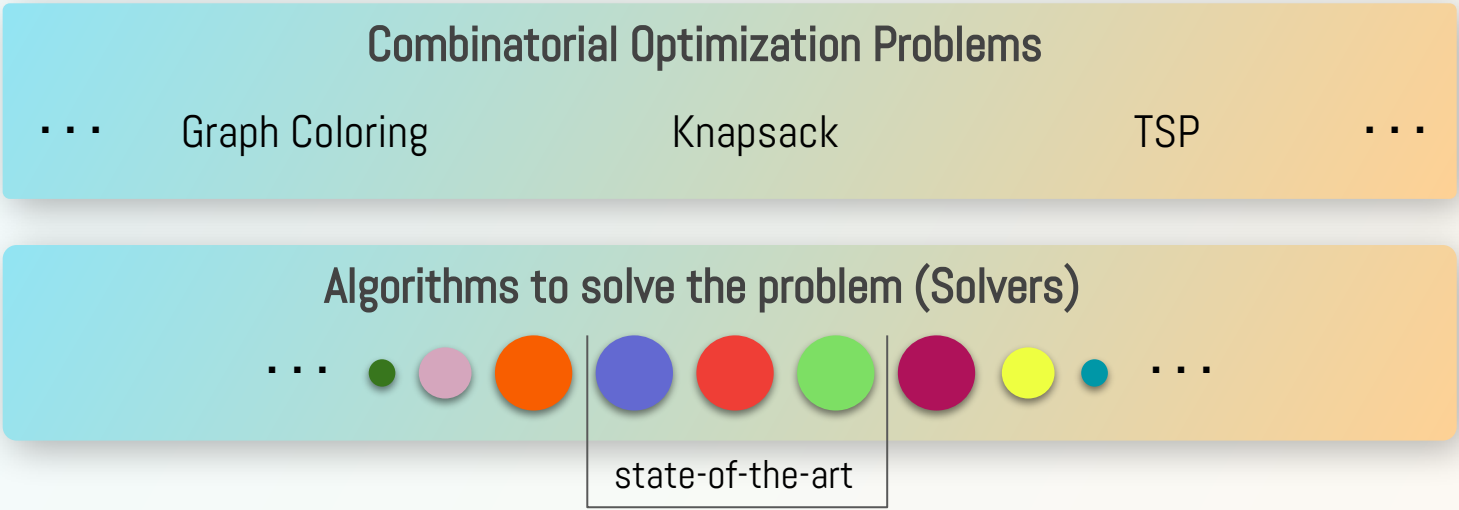
Automatic Algorithm Selection

Combinatorial Optimization Problems

... Graph Coloring Knapsack TSP ...

Algorithms to solve the problem (Solvers)

Automatic Algorithm Selection



Automatic Algorithm Selection

Combinatorial Optimization Problems

· · · Graph Coloring Knapsack TSP · · ·

State-of-the-art solvers

¿ ?

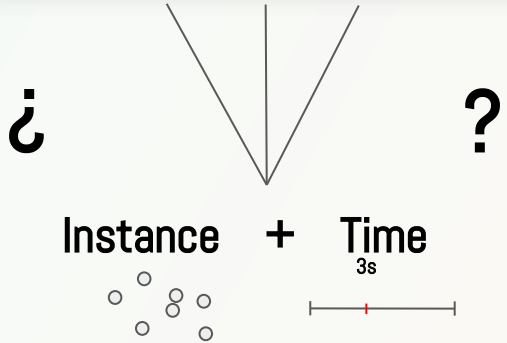
Instance

Anytime Automatic Algorithm Selection

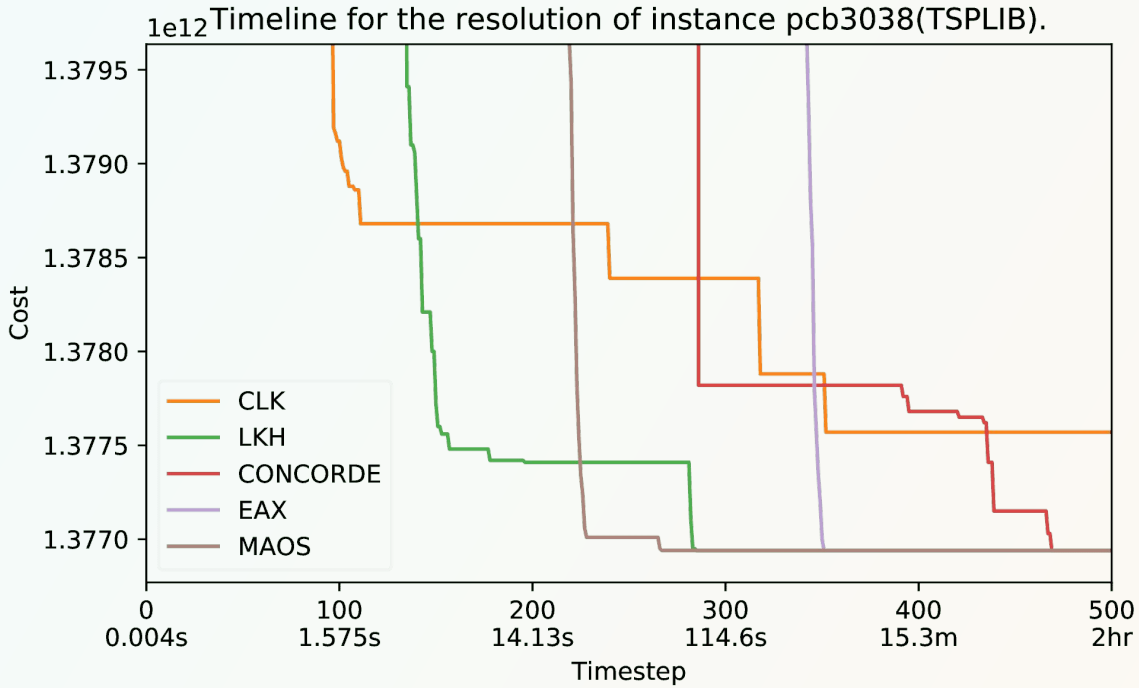
Combinatorial Optimization Problems

... Graph Coloring Knapsack TSP ...

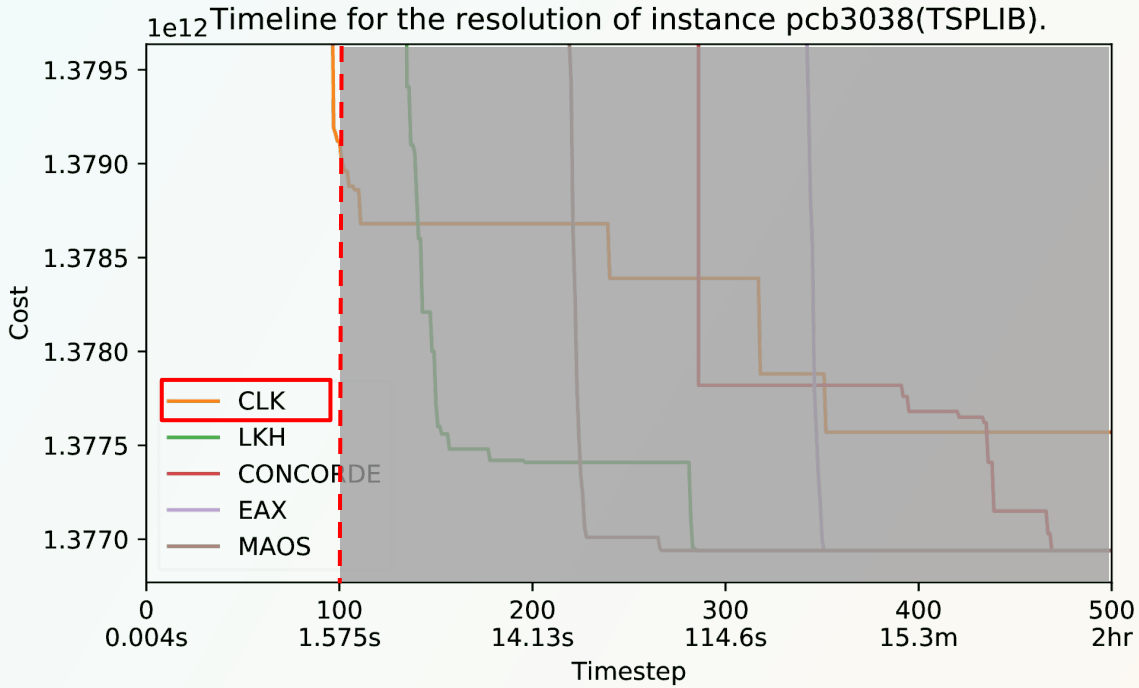
State-of-the-art solvers



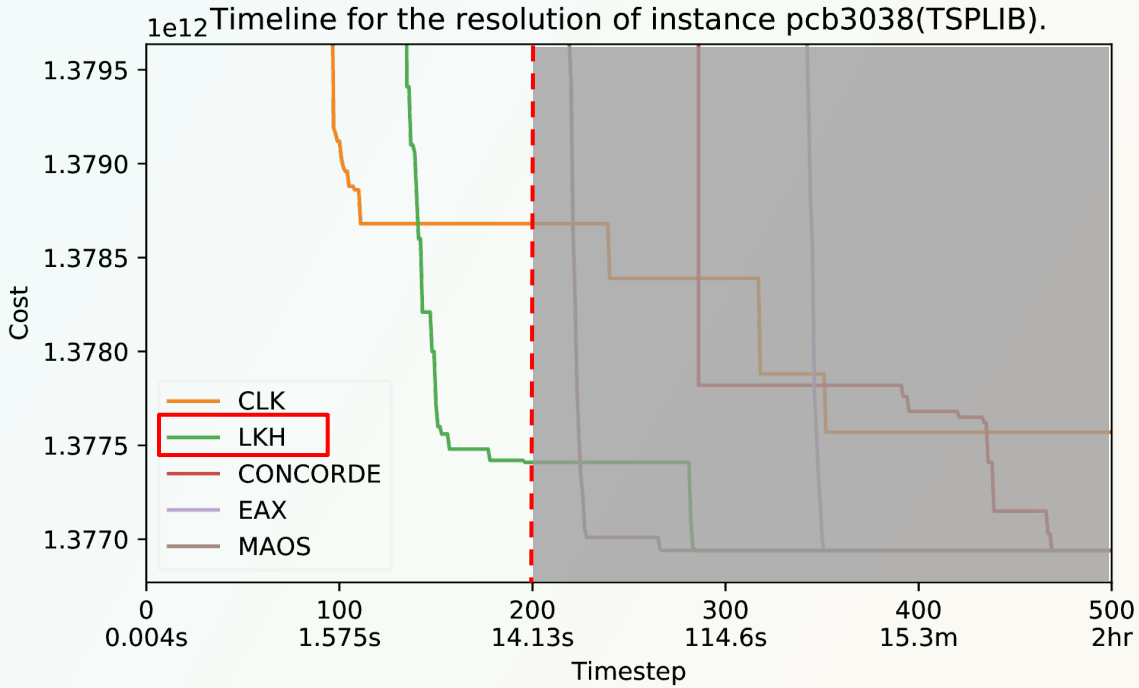
Anytime Automatic Algorithm Selection



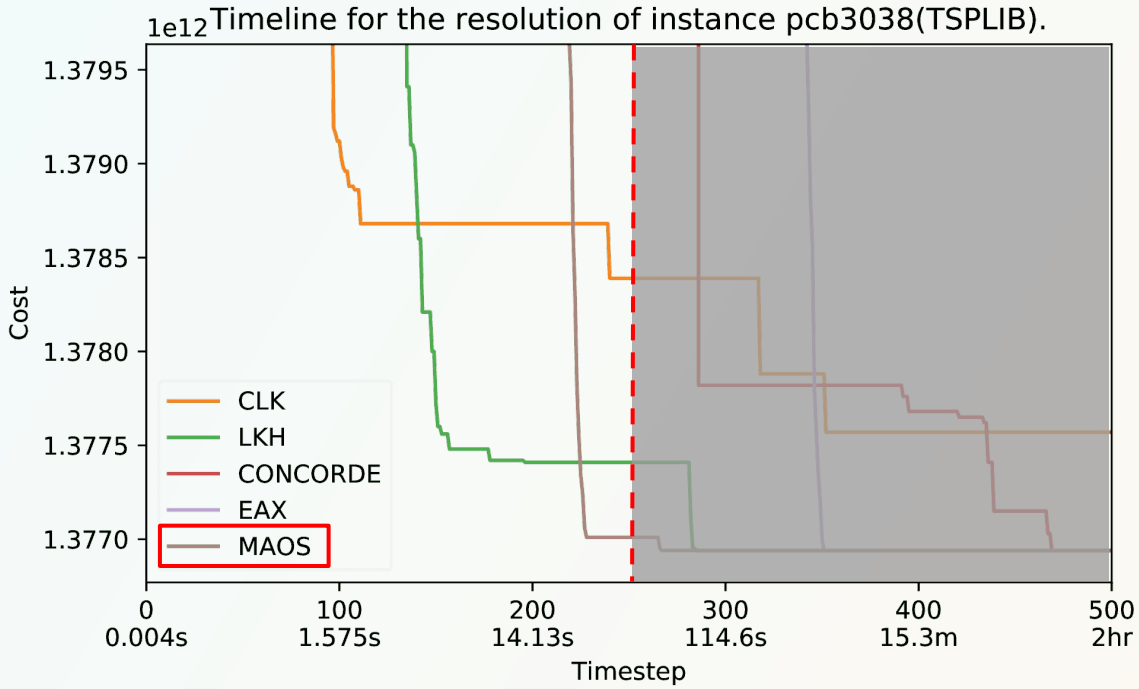
Anytime Automatic Algorithm Selection



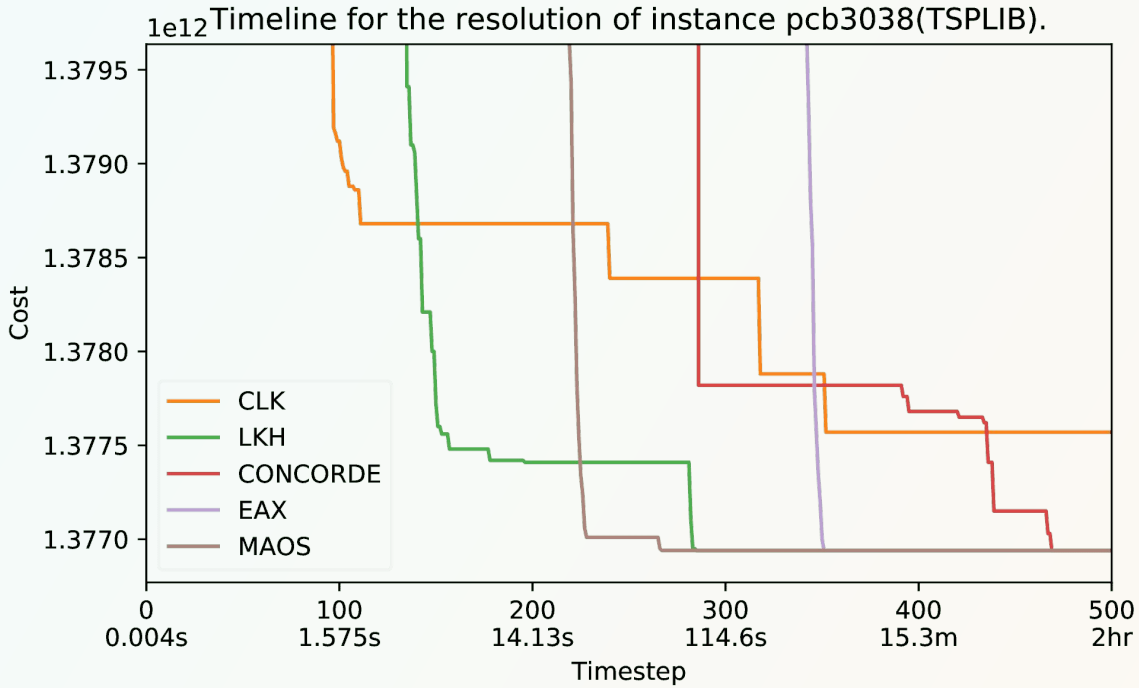
Anytime Automatic Algorithm Selection



Anytime Automatic Algorithm Selection



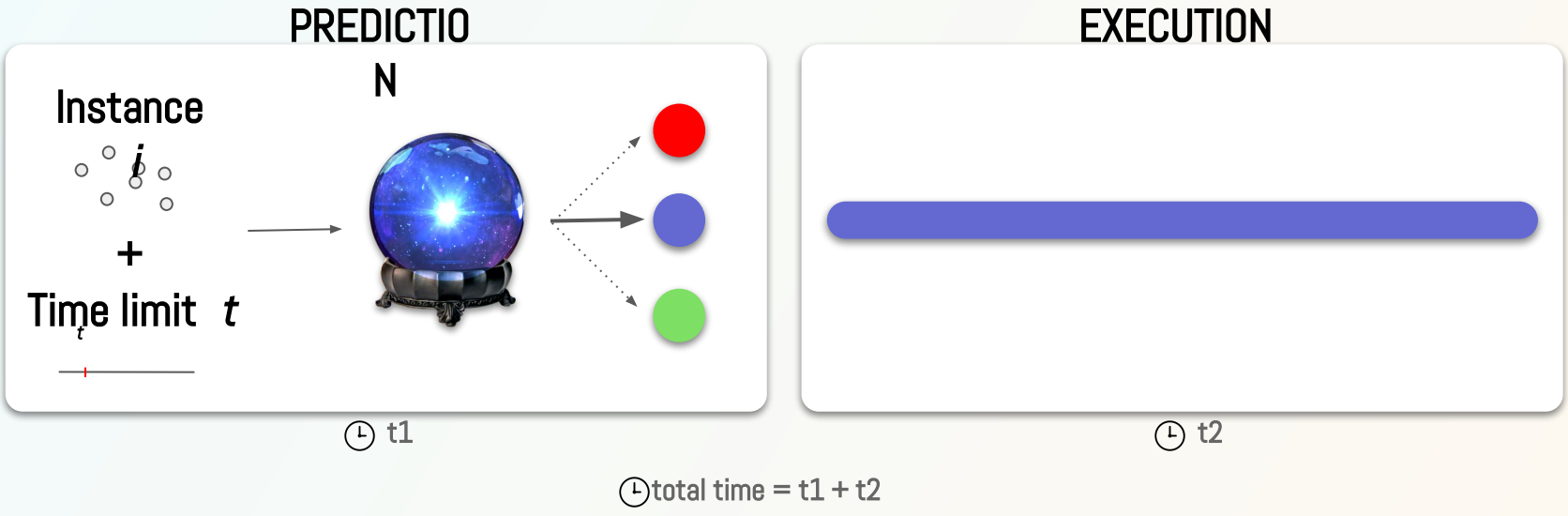
Anytime Automatic Algorithm Selection



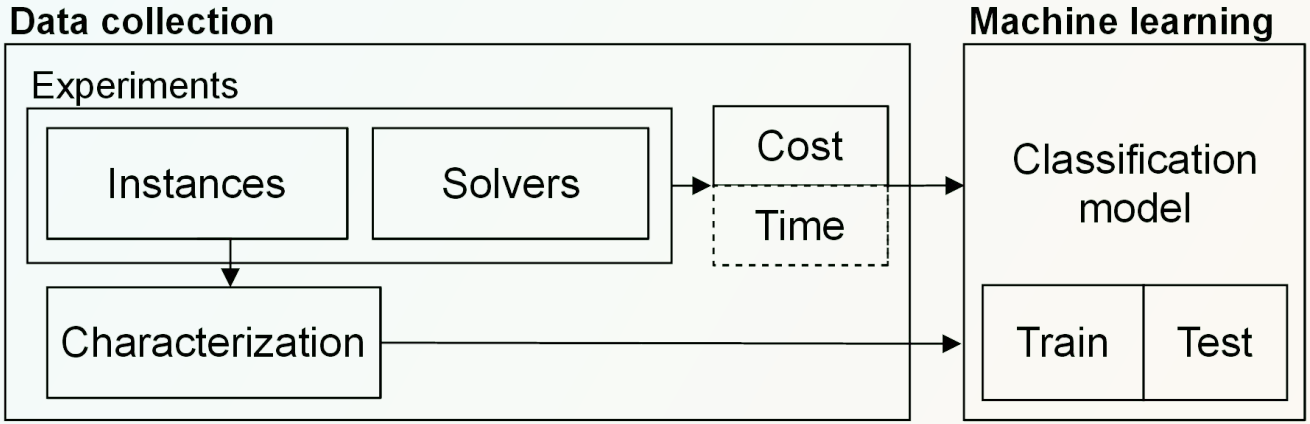
Main Goal

Build a new metasolver for Euclidean TSP based on anytime automatic algorithm selection. The main question that this metaheuristic seeks to answer is:

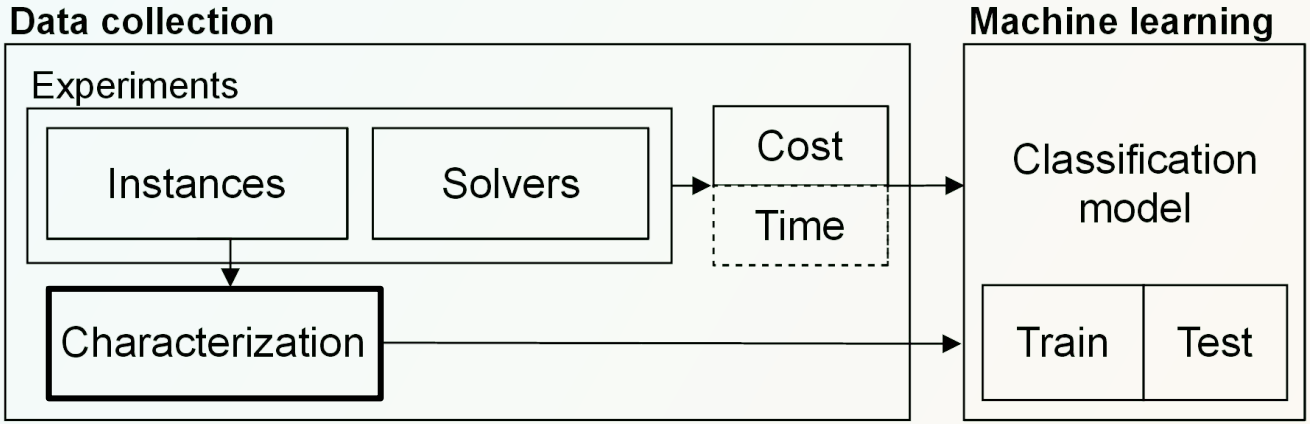
Given an **instance** *i* and a **time limit** *t*, which **solver** should be used to solve the instance i?



Automatic Algorithm Selection



Automatic Algorithm Selection



+ Features ↑ ↓ - Computational cost

TSP Instances

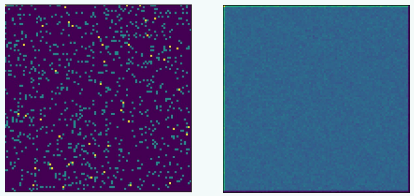
Generated Instances

Train data
6331 Instances

RUE [11]

3150 instances.

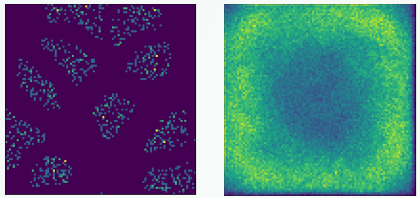
sample 1 | average



NETGENM [2]

600 instances.

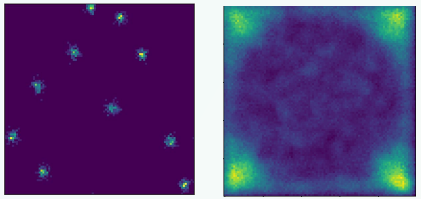
sample 1 | average



NETGEN [2]

600 instances.

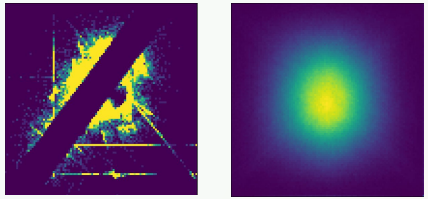
sample 1 | average



TSPGEN [12]

1981 instances.

sample 1 | average



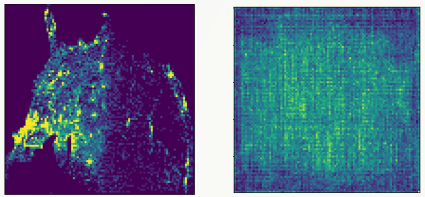
Public Instances

Test data
358 Instances

TSPLIB [8]

88 instances.

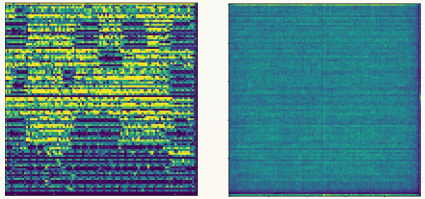
sample 1 | average



VLSI [9]

102 instances.

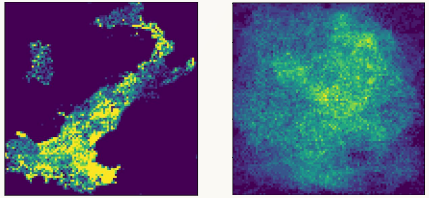
sample 1 | average



NATIONAL [9]

27 instances.

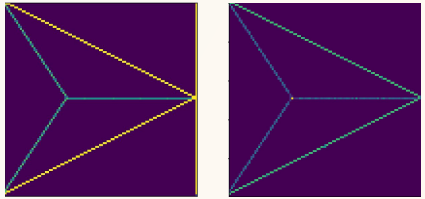
sample 1 | average



TNM [10]

141 instances.

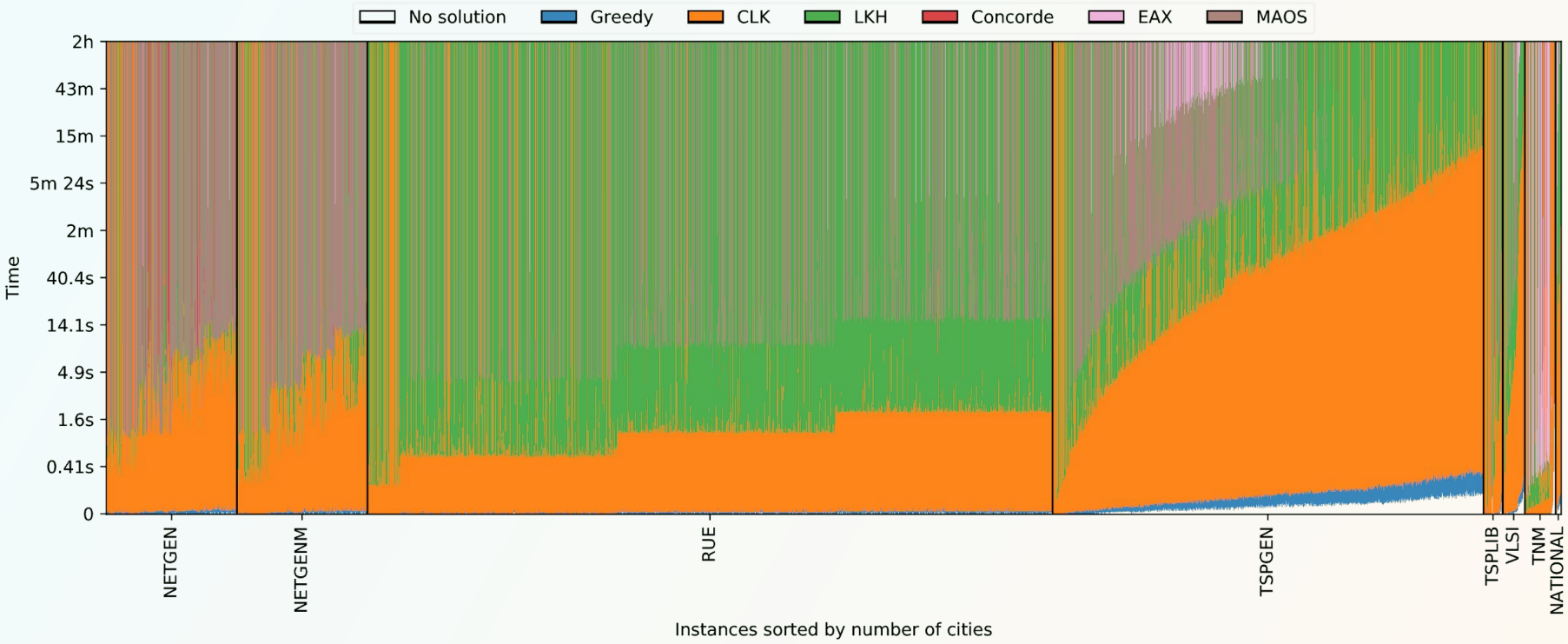
sample 1 | average



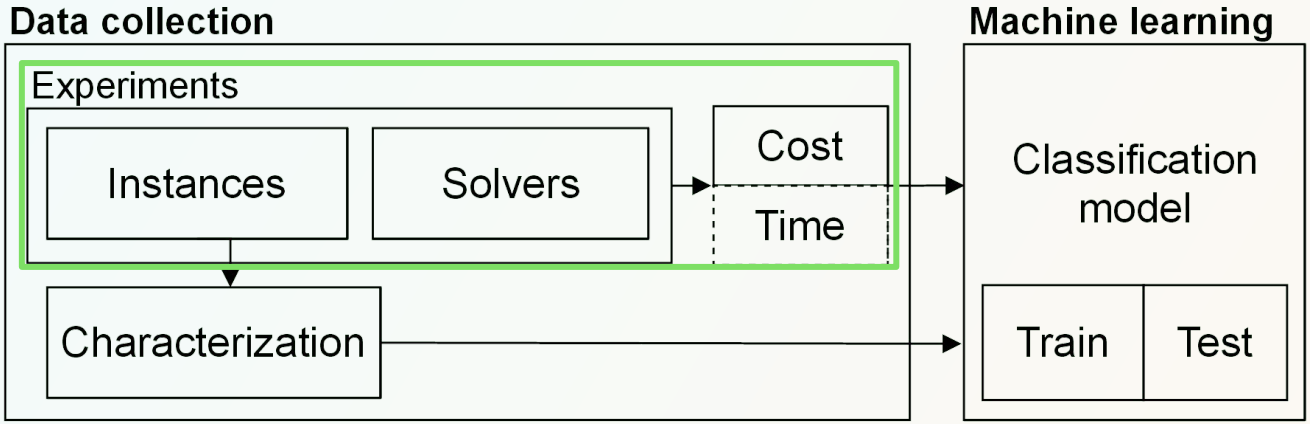
Solvers

	Name	Algorithm	Exact	Language	Year	Author
Concorde [3]	Concorde	Branch & Bound	Yes	C	2006	D. Applegate
CLK [4]	Chained Lin Kernighan	ILS	No	C	2003	D. Applegate
LKH [5]	Lin Kernighan Helsgaun	ILS	No	C	2018	Helsgaun
EAX [6]	Edge Assembly Crossover	Genetic	No	C++	2013	Nagata & Kobayashi
MAOS [7]	Multi Agent Optimization System	Cooperative search	No	Java	2009	Xie & Liu

Solver-Instance Execution



Motivation


Data collection
Experiments
Instances
Solvers
Cost
Time
Characterization
Machine learning
Classification model
Train
Test

Motivation

Characterization problem

- ➔ Costly feature calculation.
- ➔ Arbitrary selection of features.
  - ◆ Overlook relevant features.
  - ◆ Consider irrelevant features.

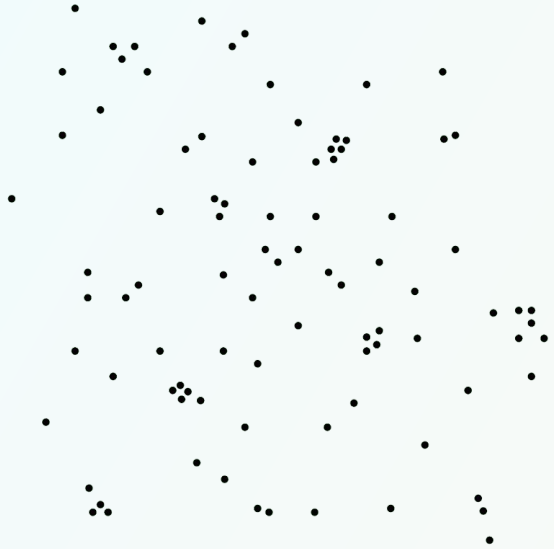
→

Proposed approach

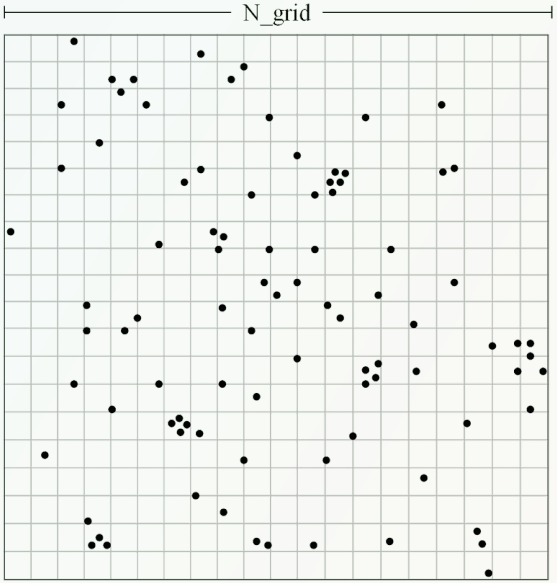
- ➔ Feature selection is avoided and replaced by automatic parameter tuning within a neural network.
- ➔ New representation suitable for the use of neural networks.

Input representation

Euclidean TSP Instance



Framing



Matrix representation

N_grid

0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0	1	0	1	5	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	1	2	0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0	1	0	0	0	2	0	0	1	0	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1
0	0	1	0	0	1	0	0	1	0	1	0	0	4	0	1	0	0	1
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	4	1	0	0	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	2	0	1	0	0	1	0	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Parameters

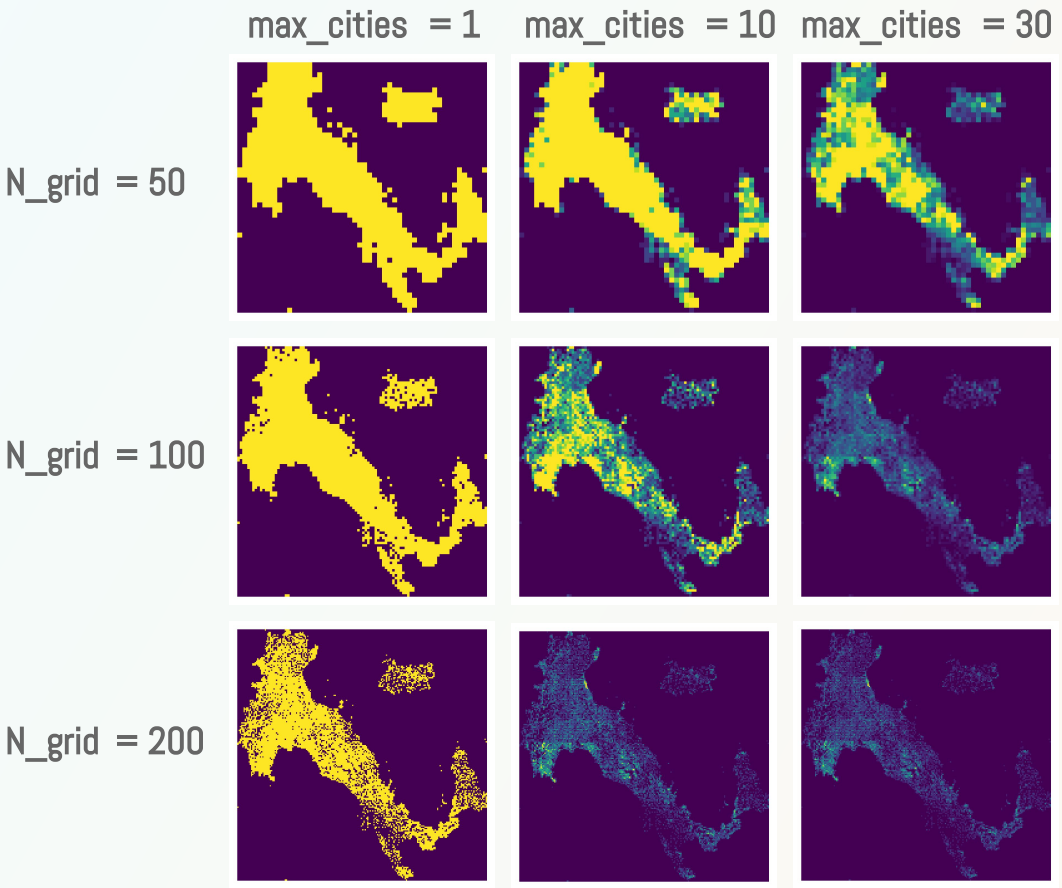
Matrix representation

N_grid

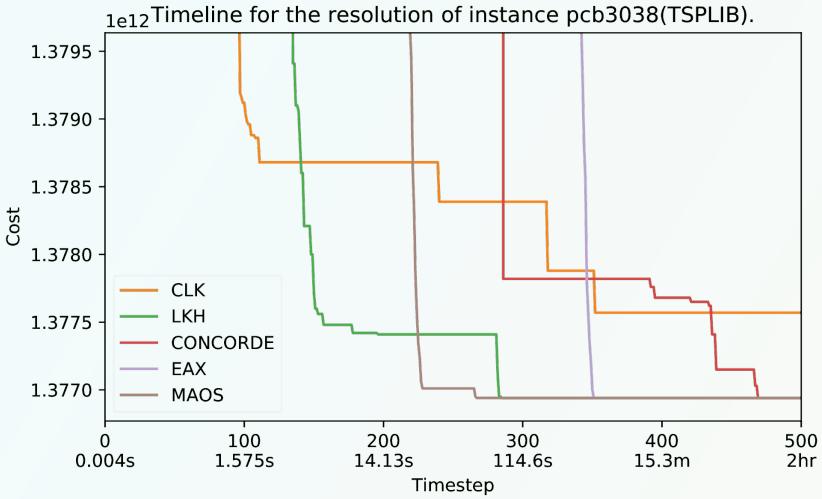
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	1	0	1	5	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	2	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	0	1	0	0	0	2	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	2
0	0	1	0	0	1	0	0	1	0	1	0	0	4	0	1	0	0	1	1
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	4	1	0	0	0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	2	0	1	0	0	1	0	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

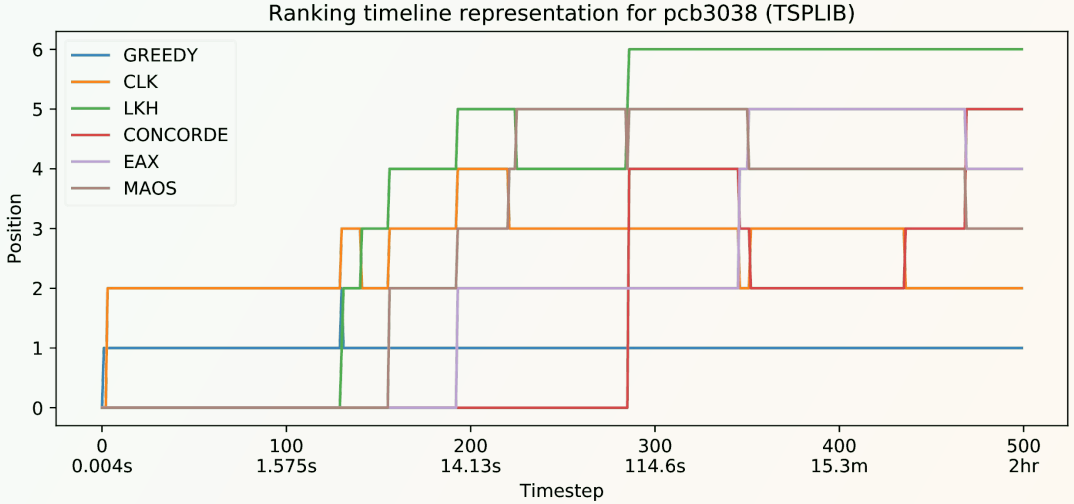
- **N_grid** : square matrix dimension.
- **max_cities** : maximum cell value. In order to normalize the matrix, the maximum value for the cells must be known.

Parameters


max_cities = 1
max_cities = 10
max_cities = 30
N_grid = 50
N_grid = 100
N_grid = 200

Output representation

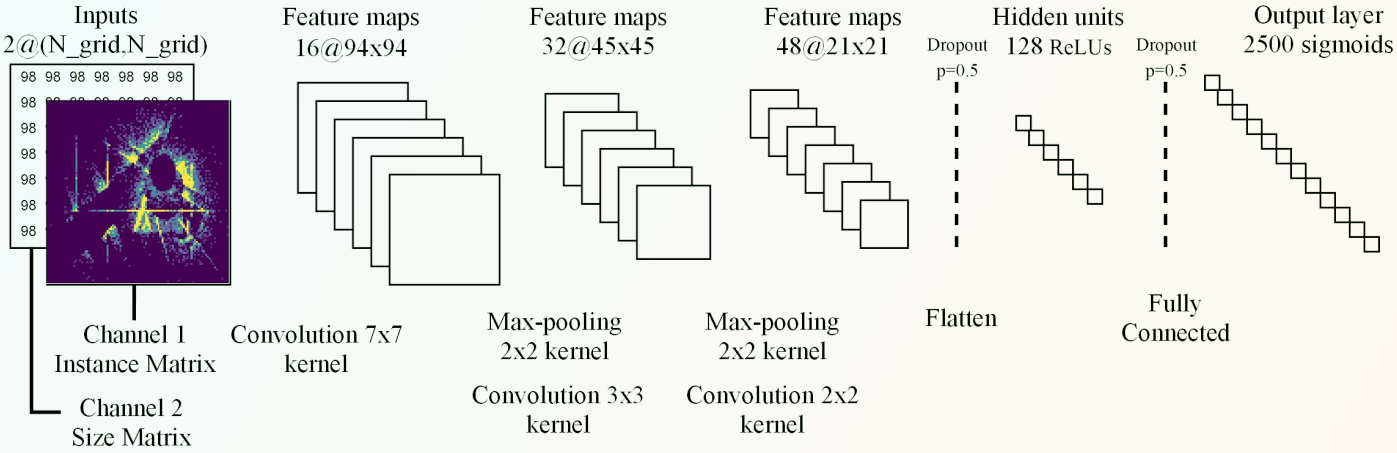




CNN Model

Convolutional Neural Network [14]

9,866,864 parameters

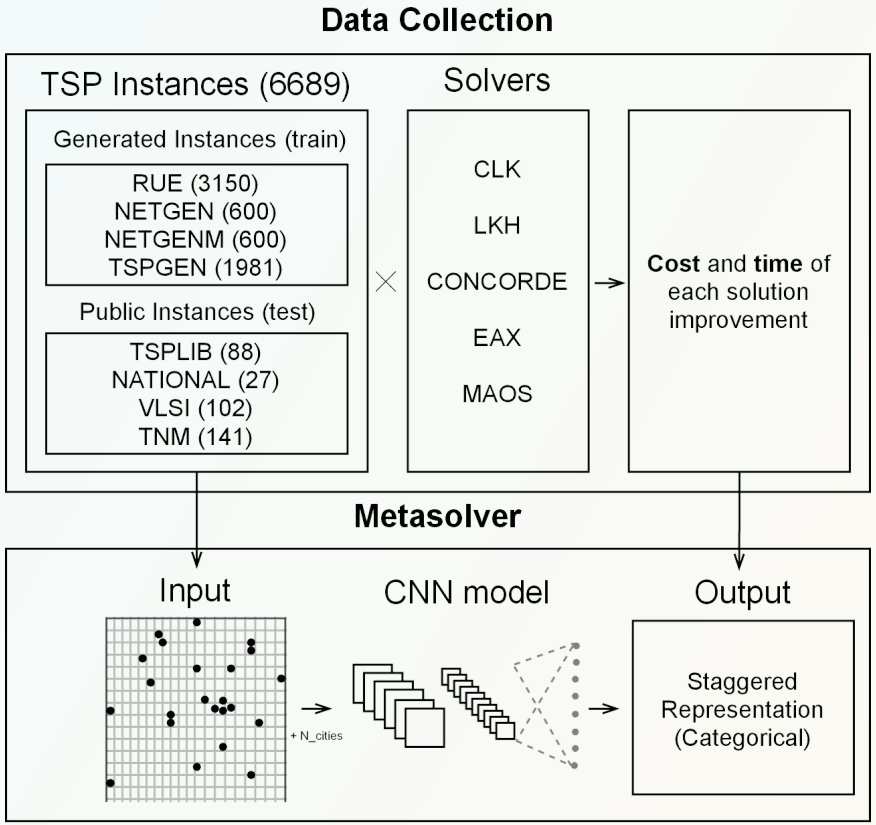


Parameters

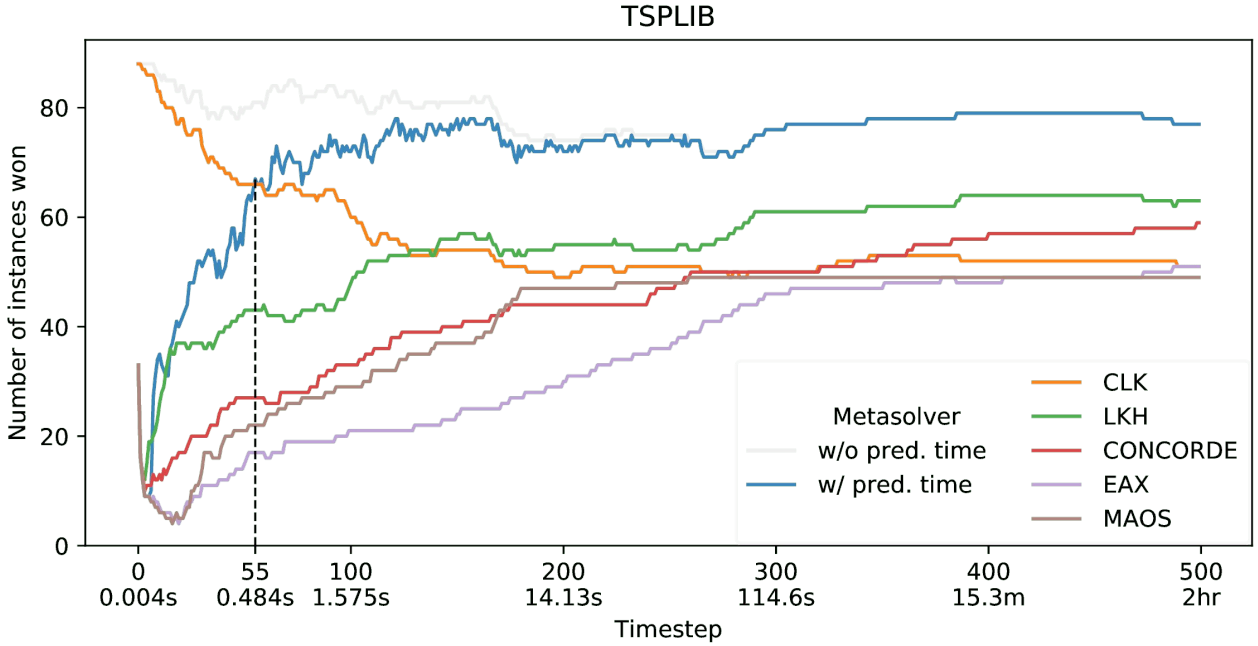
- Adam optimizer
- learning rate: 0.00001
- loss: binary cross-entropy

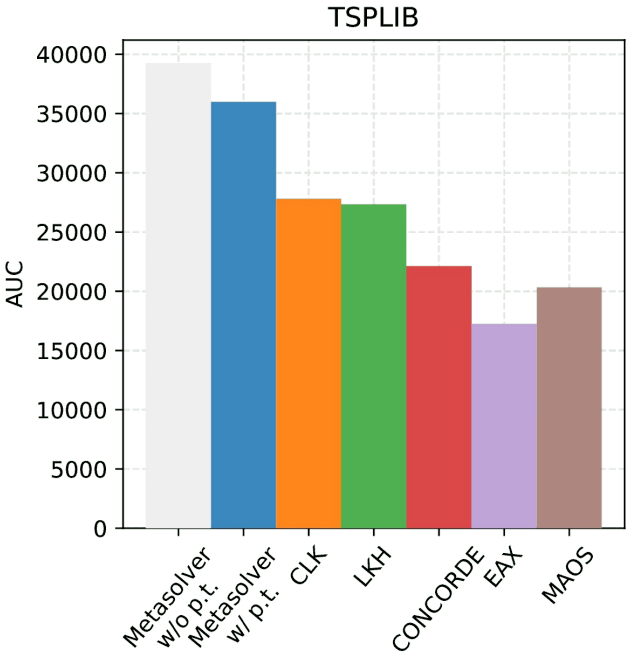
Categorical accuracy: 0.667

General Framework

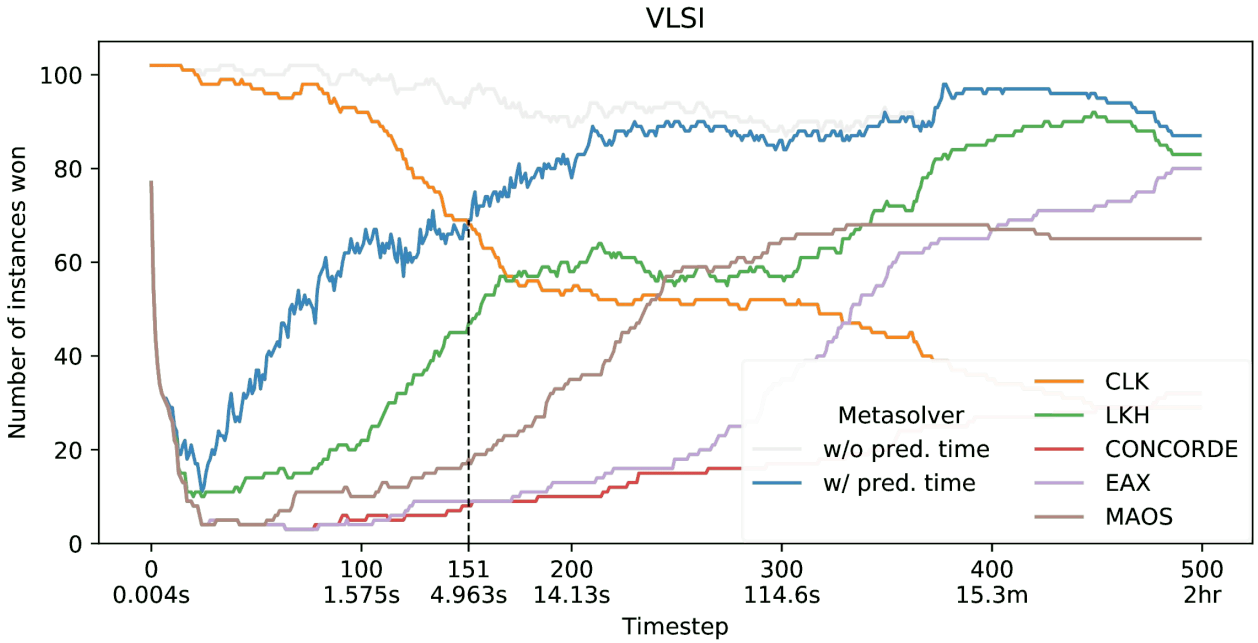


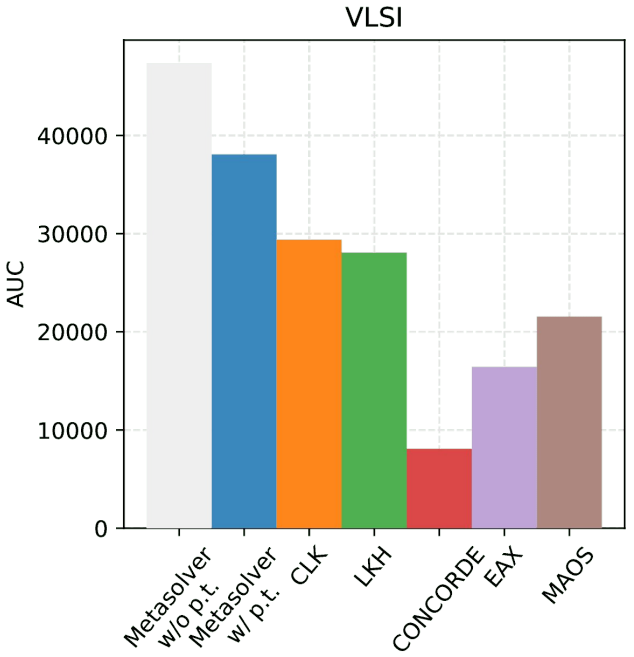
Results


TSPLIB
Number of instances won
0
20
40
60
80
0
0.004s
55
0.484s
100
1.575s
200
14.13s
300
114.6s
400
15.3m
500
2hr
Timestep
Metasolver
w/o pred. time
w/ pred. time
CLK
LKH
CONCORDE
EAX
MAOS

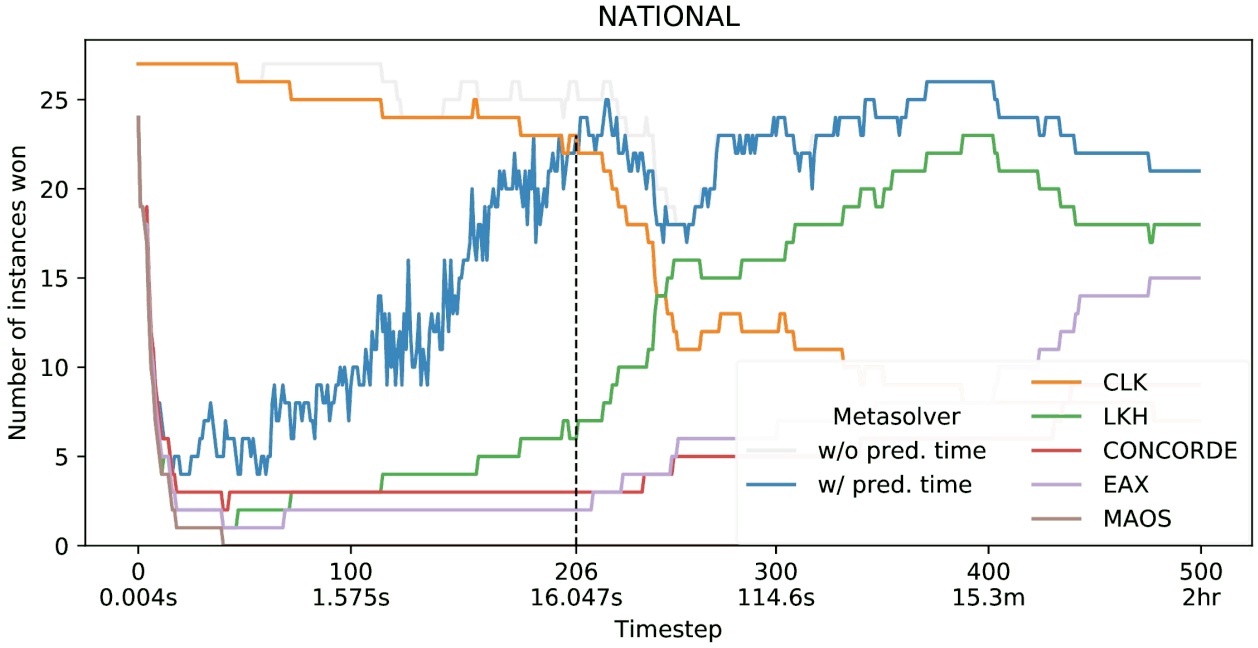

TSPLIB
AUC
0
5000
10000
15000
20000
25000
30000
35000
40000
Metasolver w/o p.t.
Metasolver w/ p.t.
CLK
LKH
CONCORDE
EAX
MAOS

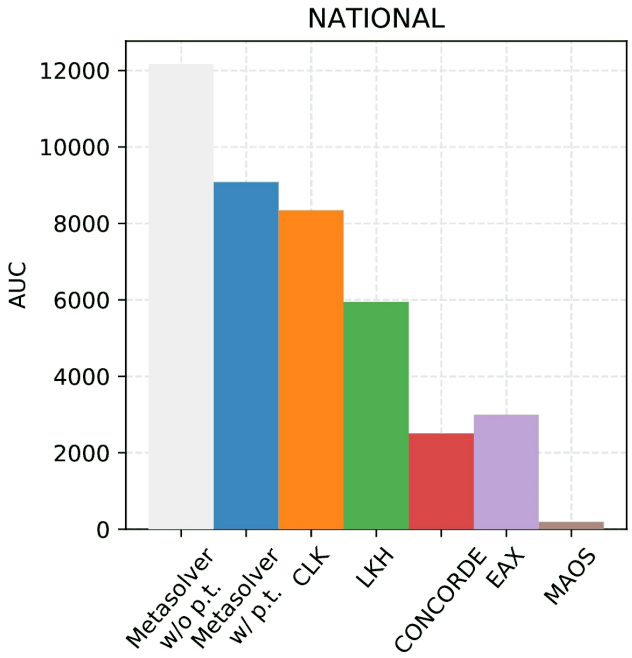
Results


VLSI
Number of instances won
100
80
60
40
20
0
0
0.004s
100
1.575s
151
4.963s
200
14.13s
300
114.6s
400
15.3m
500
2hr
Timestep
Metasolver
w/o pred. time
w/ pred. time
CLK
LKH
CONCORDE
EAX
MAOS

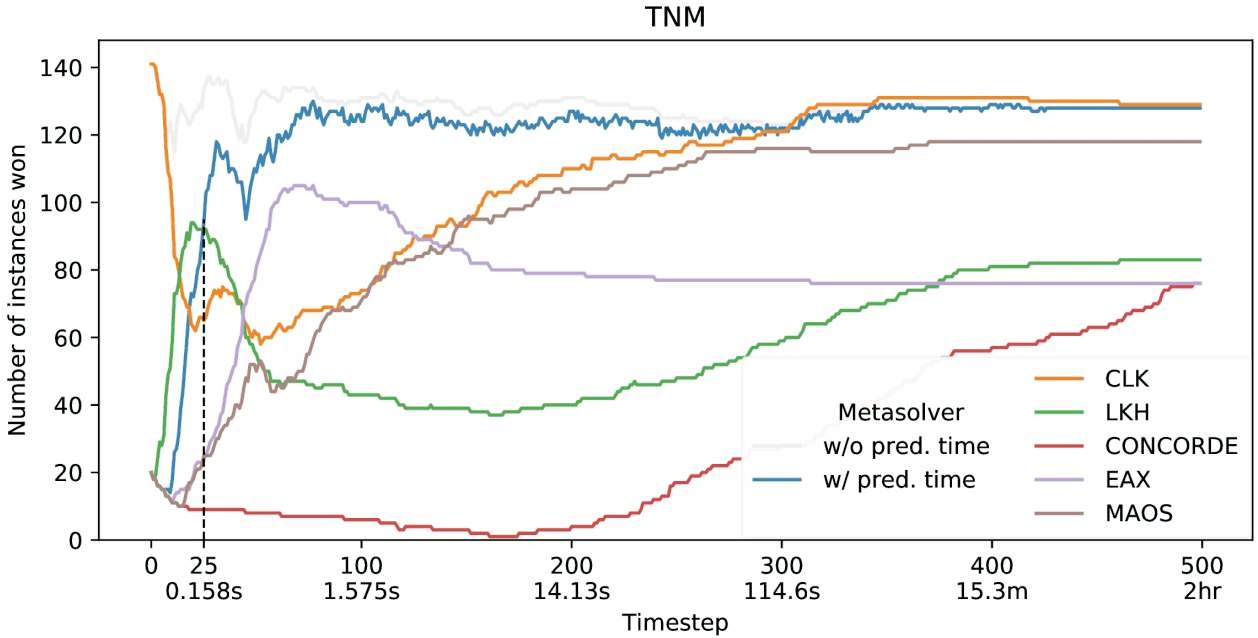

VLSI
AUC
40000
30000
20000
10000
0
Metasolver w/o p.t.
Metasolver w/ p.t.
CLK
LKH
CONCORDE
EAX
MAOS

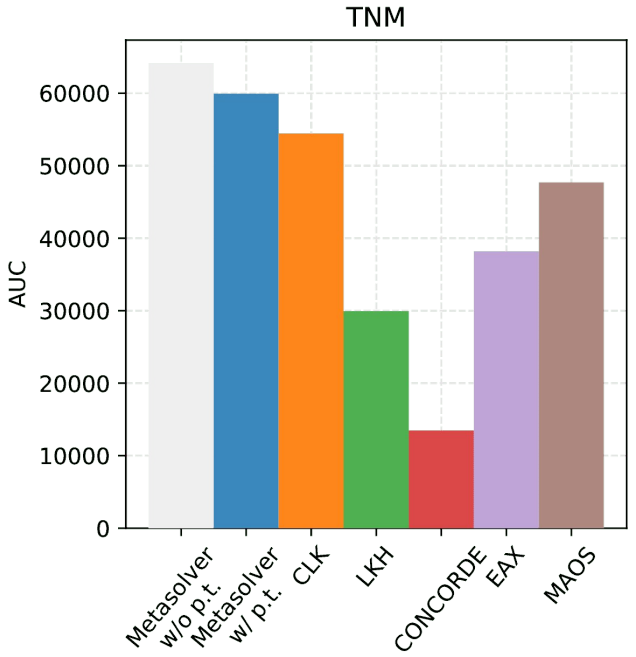
Results


NATIONAL
Number of instances won
0
5
10
15
20
25
0
0.004s
100
1.575s
206
16.047s
300
114.6s
400
15.3m
500
2hr
Timestep
Metasolver
w/o pred. time
w/ pred. time
CLK
LKH
CONCORDE
EAX
MAOS


NATIONAL
AUC
0
2000
4000
6000
8000
10000
12000
Metasolver w/o p.t.
Metasolver w/ p.t.
CLK
LKH
CONCORDE
EAX
MAOS

Results


TNM
Number of instances won
0
20
40
60
80
100
120
140
0
25
0.158s
100
1.575s
200
14.13s
300
114.6s
400
15.3m
500
2hr
Timestep
Metasolver
w/o pred. time
w/ pred. time
CLK
LKH
CONCORDE
EAX
MAOS


TNM
AUC
0
10000
20000
30000
40000
50000
60000
Metasolver w/o p.t.
Metasolver w/ p.t.
CLK
LKH
CONCORDE
EAX
MAOS

Results summary

$$Accuracy(s, ds) = \frac{AUC(s,ds)}{500 \cdot size(ds)}, \qquad \begin{array}{l} s : solver \\ ds : dataset \end{array}$$

Accuracy (%)	TSPLIB	VLSI	NATIONAL	TNM	ALL
Metasolver (without prediction time)	90.0	91.3	89.5	90.8	90.7
Metasolver (with prediction time)	**82.7**	**73.2**	**66.9**	**85.1**	**79.8**
CLK	63.1	57.5	61.8	77.2	67.0
LKH	62.1	55.0	44.0	42.4	51.0
CONCORDE	50.2	15.8	18.5	19.1	25.8
EAX	39.2	32.2	22.2	54.1	41.8
MAOS	46.2	42.2	01.4	67.6	50.1

Conclusions

- ➔ New model for automatic algorithm selection which surpasses the performance of state-of-the-art solvers in all the considered public datasets.
- ➔ **Anytime** predictions.
- ➔ Fast predictions (0.02s vs 9s on average).
- ➔ 6,689 instances and 2 hours resolutions (isaiash.github.io/anytime_tsp).

Future work

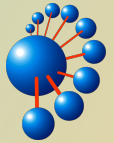
- ➔ Explore the idea of mapping the times to other systems.
- ➔ The anytime idea can be extended to other computational resources.

References

1. Huerta, I. I., Neira, D. A., Ortega, D. A., Varas, V., Godoy, J., & Asín-Achá, R. (2020). Anytime automatic algorithm selection for knapsack. Expert Systems with Applications, (p. 113613).
2. Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., & Trautmann, H. (2018). Leveraging tsp solver complementarity through machine learning.
3. Applegate, D., Cook, W., R Bixby (2006). Concorde tsp solver.
4. Applegate, D., Cook, W., & Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems.
5. Helsgaun, K. (2018). Using popmusic for candidate set generation in the lin-kernighan-helsgaun tsp solver.
6. Nagata, Y., & Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem.
7. Xie, X.-F., & Liu, J. (2008). Multiagent optimization system for solving the traveling salesman problem.

References

8. Gerhard Reinelt. (1991). TSPLIB—A Traveling Salesman Problem Library.
9. http://www.math.uwaterloo.ca/tsp/
10. Stefan Hougardy and Xianghui Zhong. (2018). Hard to Solve Instances of the Euclidean Traveling Salesman Problem.
11. http://dimacs.rutgers.edu/archive/Challenges/TSP/download.html
12. Bossek, J., Kerschke, P., Neumann, A., Wagner, M., Neumann, F., & Trautmann, H. (2019). Evolving diverse tsp instances by means of novel and creative mutation operators.
13. https://slurm.schedmd.com/documentation.html
14. Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition.



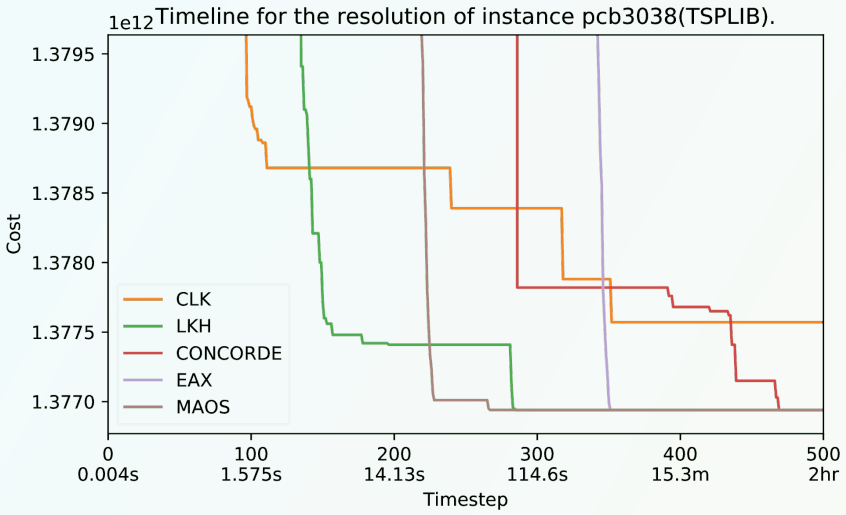
Improving the state-of-the-art in the Traveling Salesman Problem: An Anytime Automatic Algorithm Selection

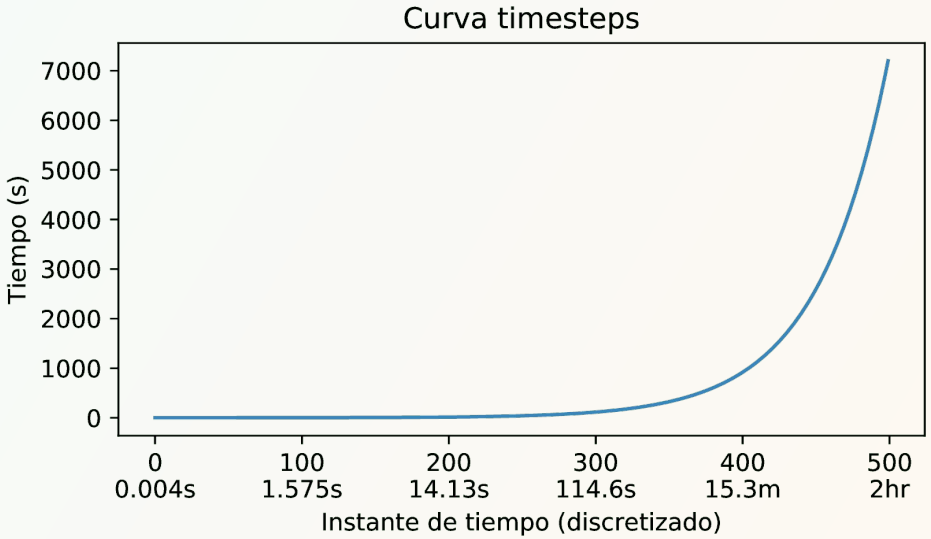
Isaías Huerta, Roberto Asín-Achá,
Julio Godoy, Daniel Neira, Daniel Ortega, Vicente Varas

Expert Systems with Applications

Universidad de Concepción | August, 26

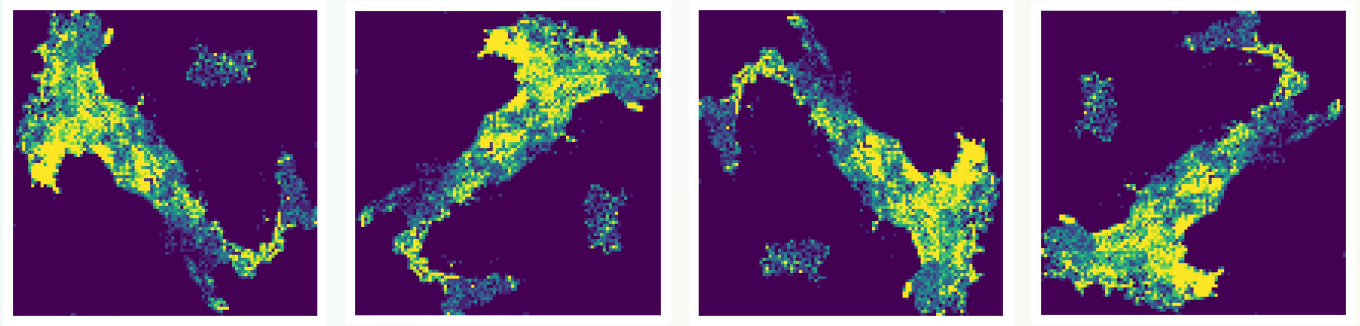
Timesteps


Timeline for the resolution of instance pcb3038(TSPLIB).
1e12
1.3795
1.3790
1.3785
1.3780
1.3775
1.3770
Cost
CLK
LKH
CONCORDE
EAX
MAOS
0
0.004s
100
1.575s
200
14.13s
300
114.6s
400
15.3m
500
2hr
Timestep


Curva timesteps
Tiempo (s)
7000
6000
5000
4000
3000
2000
1000
0
0
0.004s
100
1.575s
200
14.13s
300
114.6s
400
15.3m
500
2hr
Instante de tiempo (discretizado)

Data Augmentation

To augment the training set, three rotations of each matrix representation were added, increasing from 6,331 to **25,324** training instances

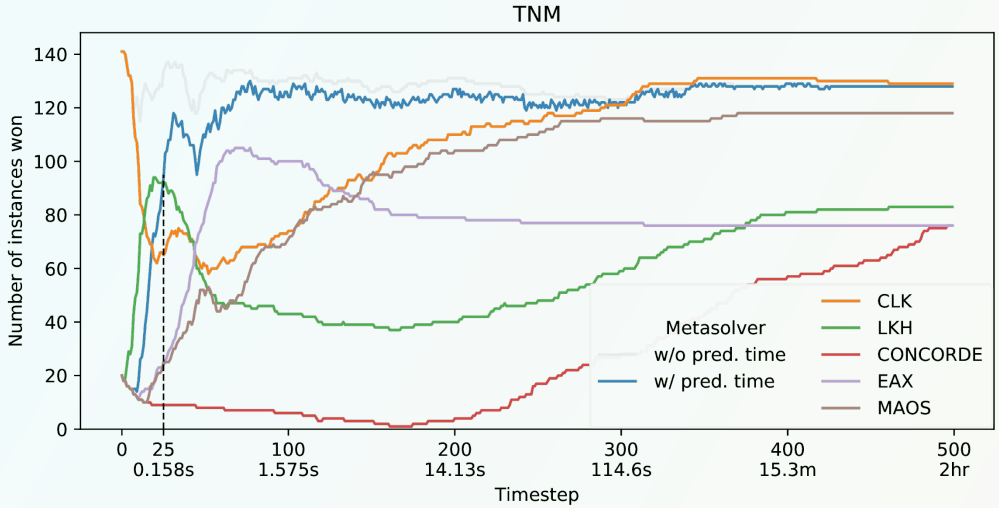


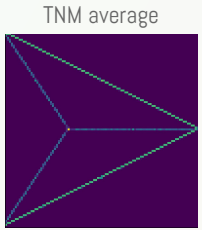
The resolution for each rotated instance is shared, so the output is repeated.

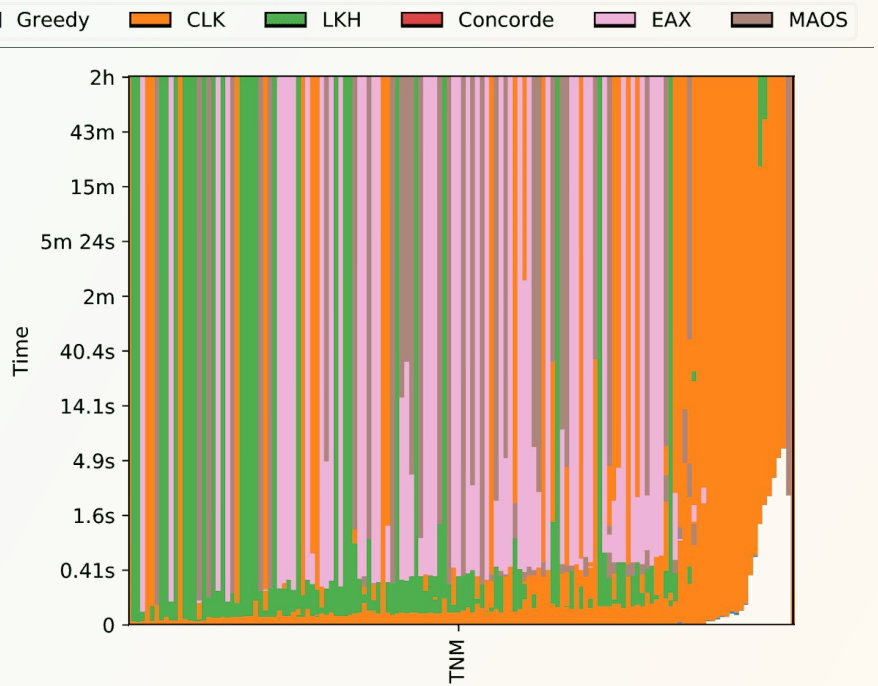
TSP Instances

	Dataset	N instances	Mean N cities	Min N cities	Max N cities	Std dev. cities	25% cities	50% cities	75% cities
Test	TSPLIB	88	3,041	14	85,900	10,185	134	428	1,468
	NATIONAL	27	11,088	29	71,009	14,558	1,800	8,079	12,412
	VLSI	102	25,654	131	744,710	90,865	1,926	3,660	19,373
	TNM	141	4,432	52	100,000	16,041	157	262	367
Train	RUE	3,150	1,452	500	2,000	451	1,000	1,500	2,000
	NETGEN	600	1,250	500	2,000	559	875	1,250	1,625
	NETGENM	600	1,250	500	2,000	559	875	1,250	1,625
	TSPGEN	1,981	10,976	11	32,304	7,373	4,969	10,004	15,783

TNM and CLK

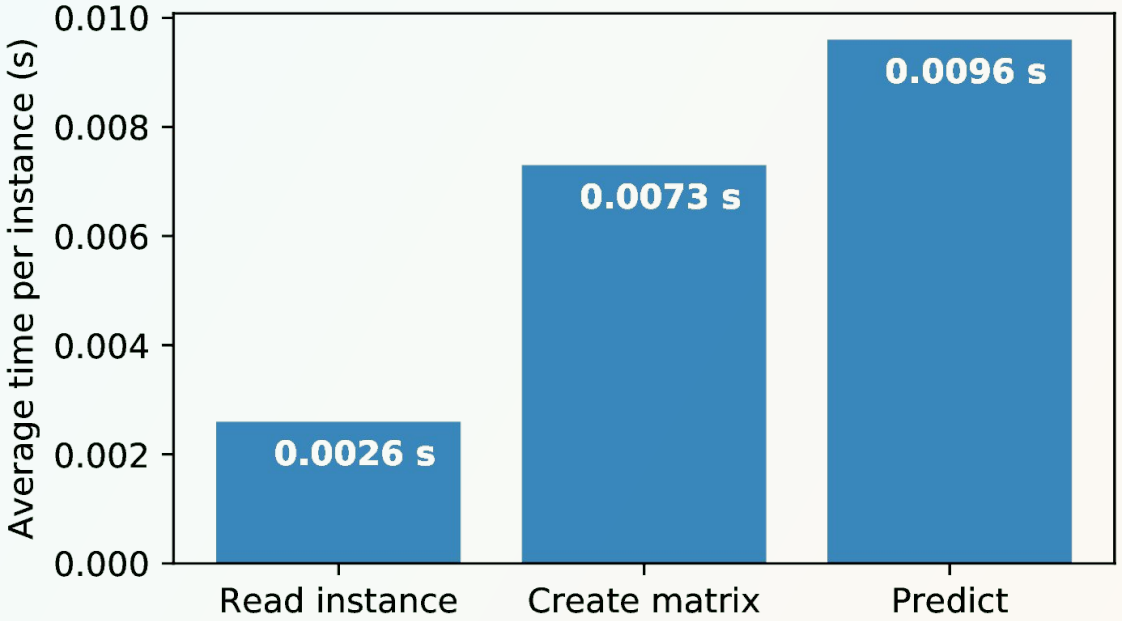

TNM
Number of instances won
Timestep
Metasolver
w/o pred. time
w/ pred. time
CLK
LKH
CONCORDE
EAX
MAOS
0
20
40
60
80
100
120
140
0
0.158s
25
1.575s
100
14.13s
200
114.6s
300
15.3m
400
2hr
500


TNM average


No solution
Greedy
CLK
LKH
Concorde
EAX
MAOS
Time
2h
43m
15m
5m 24s
2m
40.4s
14.1s
4.9s
1.6s
0.41s
0
TNM

Time measure for prediction

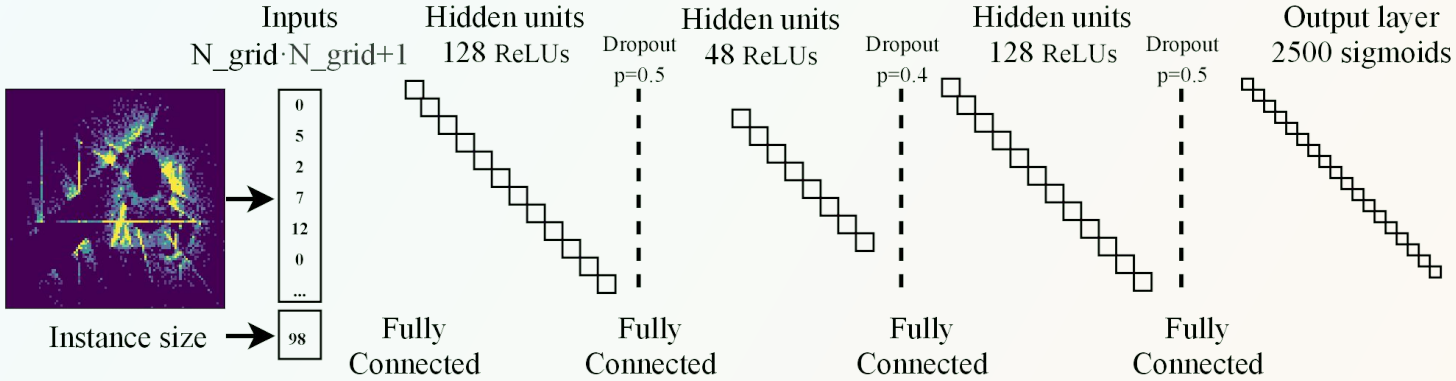
Average prediction time: 0.02s.



Models

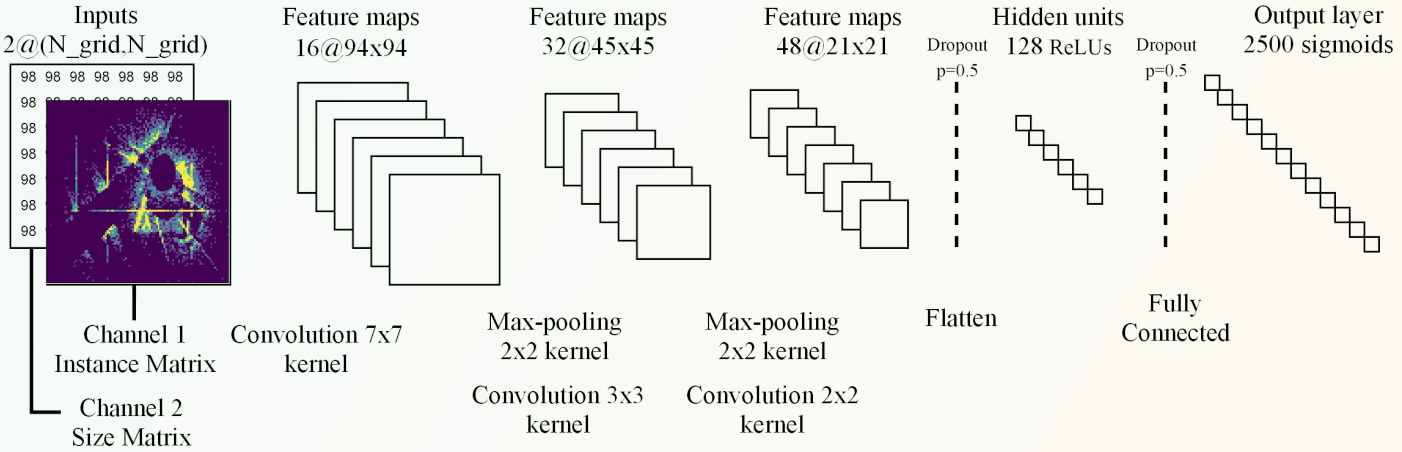
Fully Connected

5,162,720 parameters



Convolutional Neural Network [14]

9,866,864 parameters

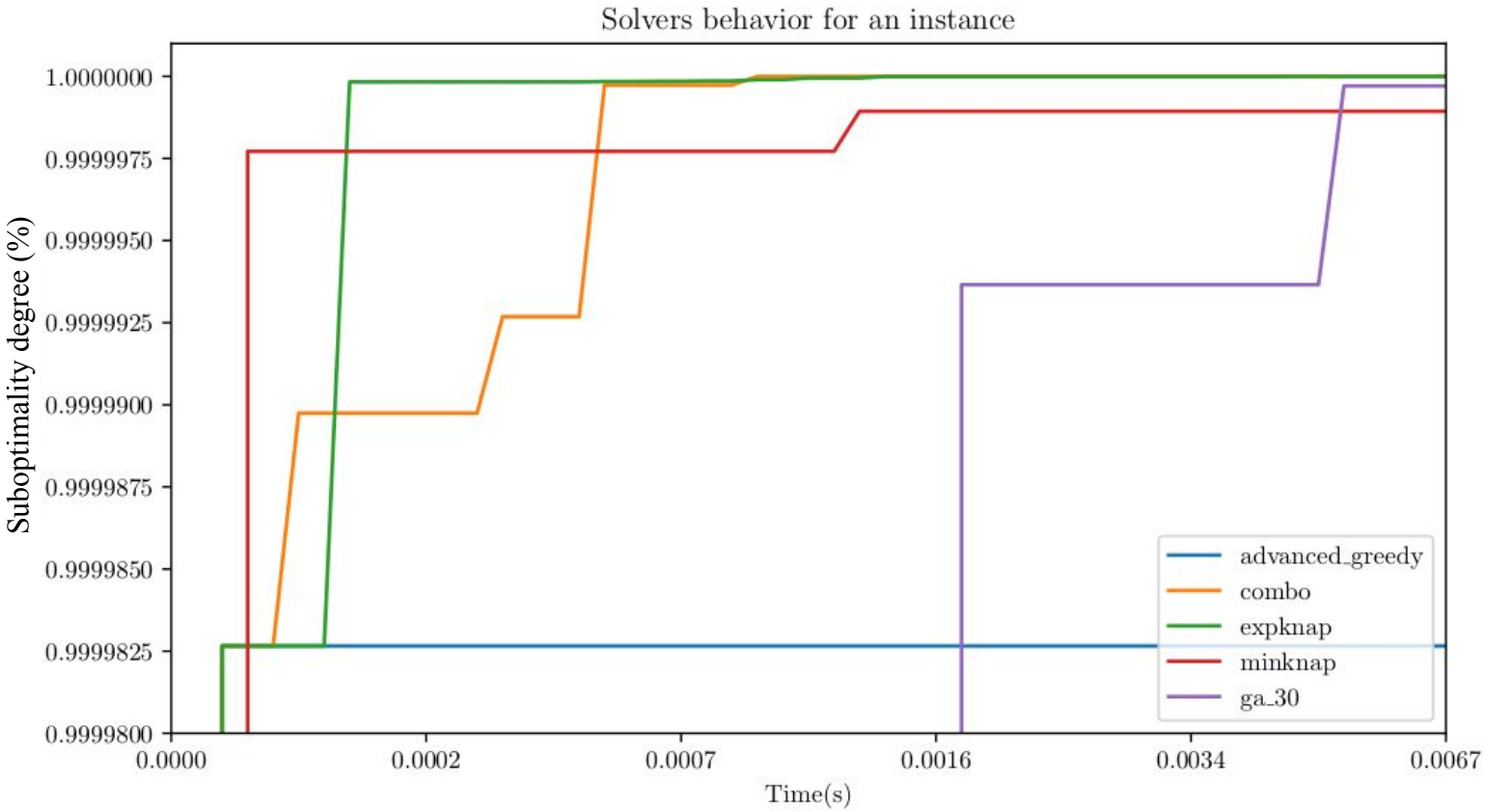


Automatic Algorithm Selection for TSP^2

Leveraging TSP Solver Complementarity through Machine Learning (Kerschke et al., 2018).

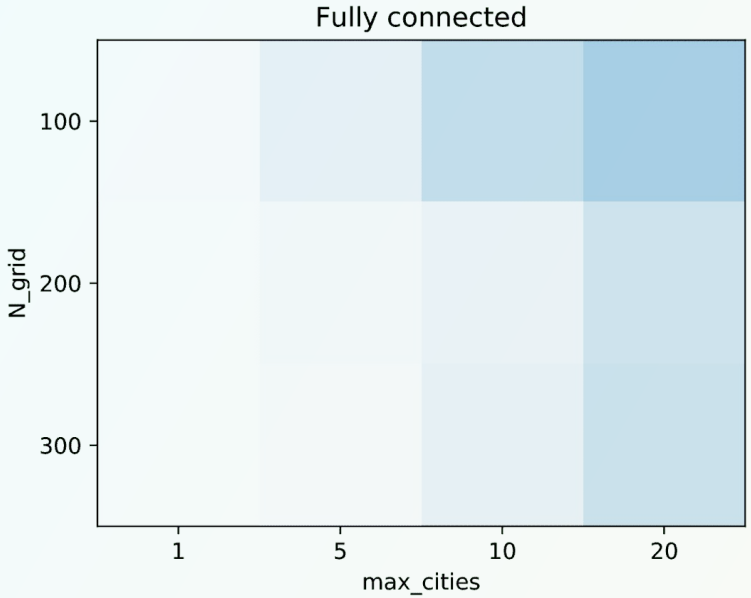
- ➔ 5 Inexact Solvers: LKH, EAX, MAOS, EAX+restart and LKH+restart.
- ➔ Instances: TSPLIB, VLSI, NATIONAL, RUE, NETGEN, MORPHED (1,845 instances).
- ➔ Cutoff time of 1 hour.
- ➔ Results: The model beats the best solver in 3 out of 6 datasets.

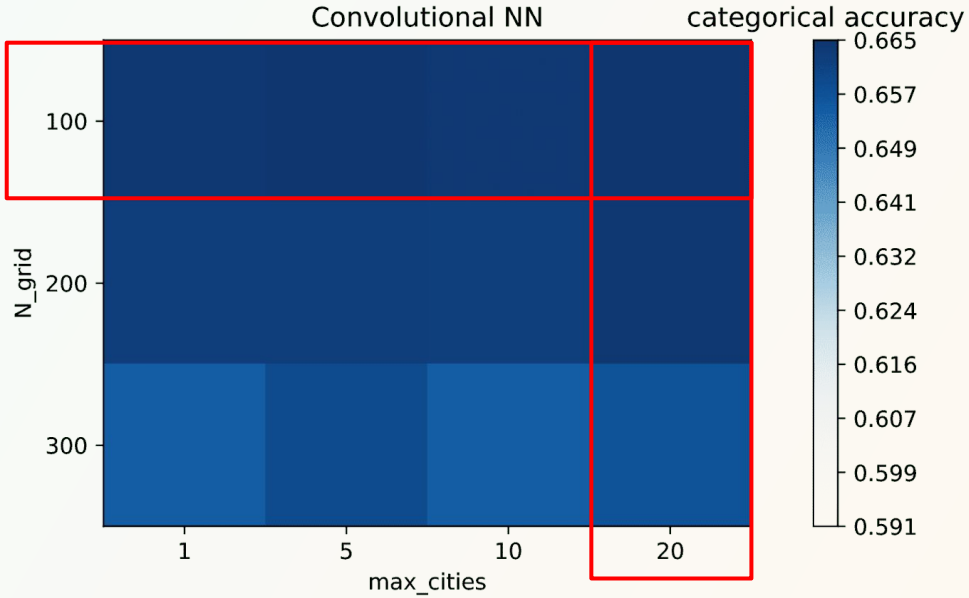
Anytime AAS for Knapsack[1]



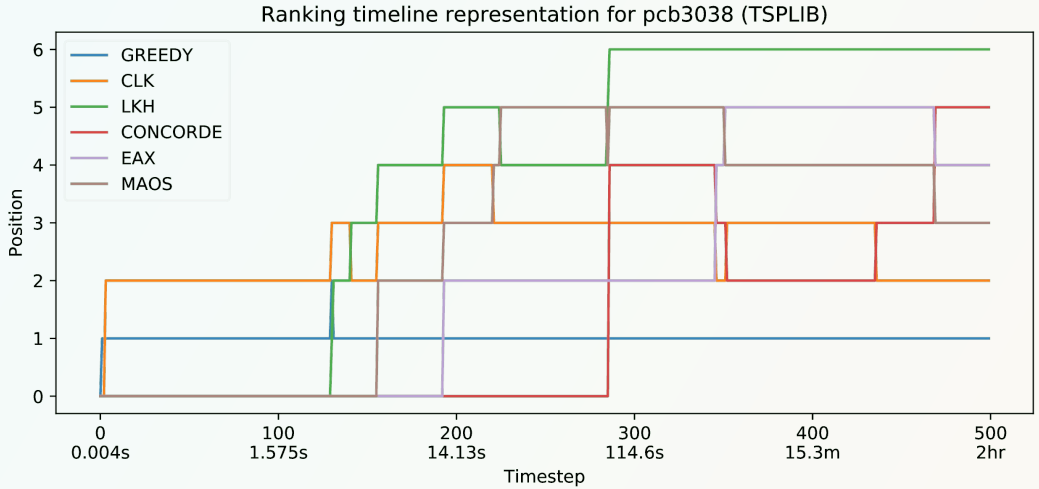
Parameter tuning

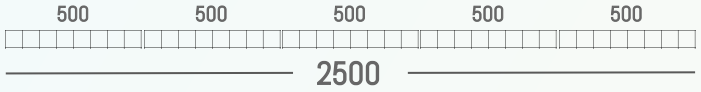
Validation data: 30% of training data (10,033 instances)



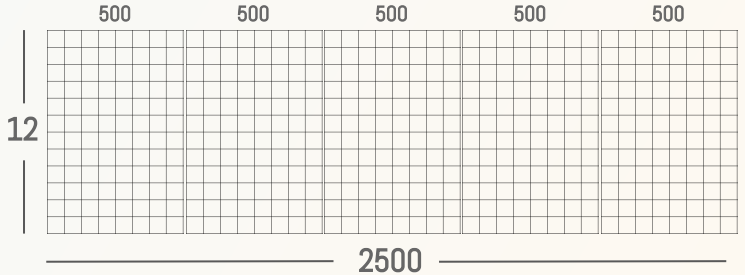


Staggered representation





One-hot encoding →



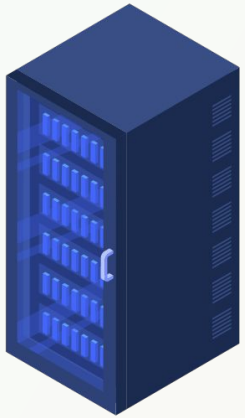
(Integer values from 0 to 12)

(Binary values)

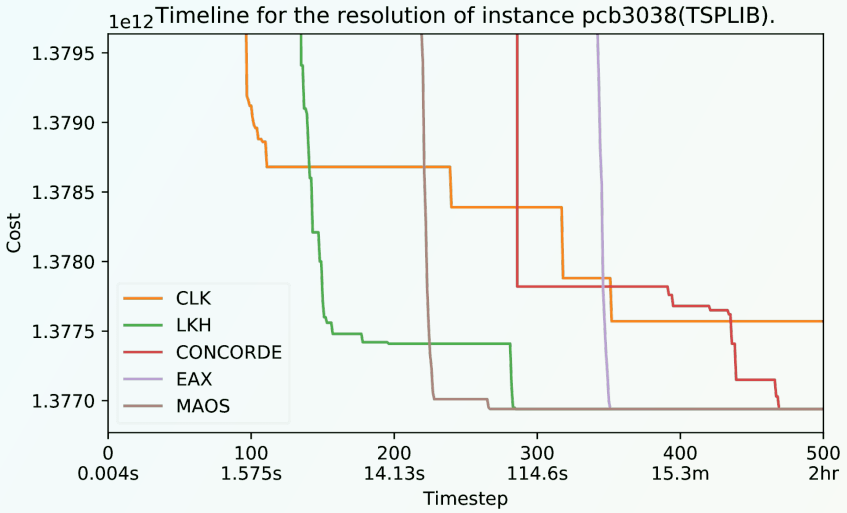
Experimental Setup

The execution of the instances with each solver was performed on **Luthier** cluster, managed with **slurm** [13]. Each cluster node has a 3.80GHz Intel Xeon E3-1270 v6 processor with 64GB of RAM and Ubuntu 18.04 distribution.

The instances were executed once per solver and with time limit of **two hours** . A timeout condition was defined when the solver does not find a new solution in a span of **10 minutes** .



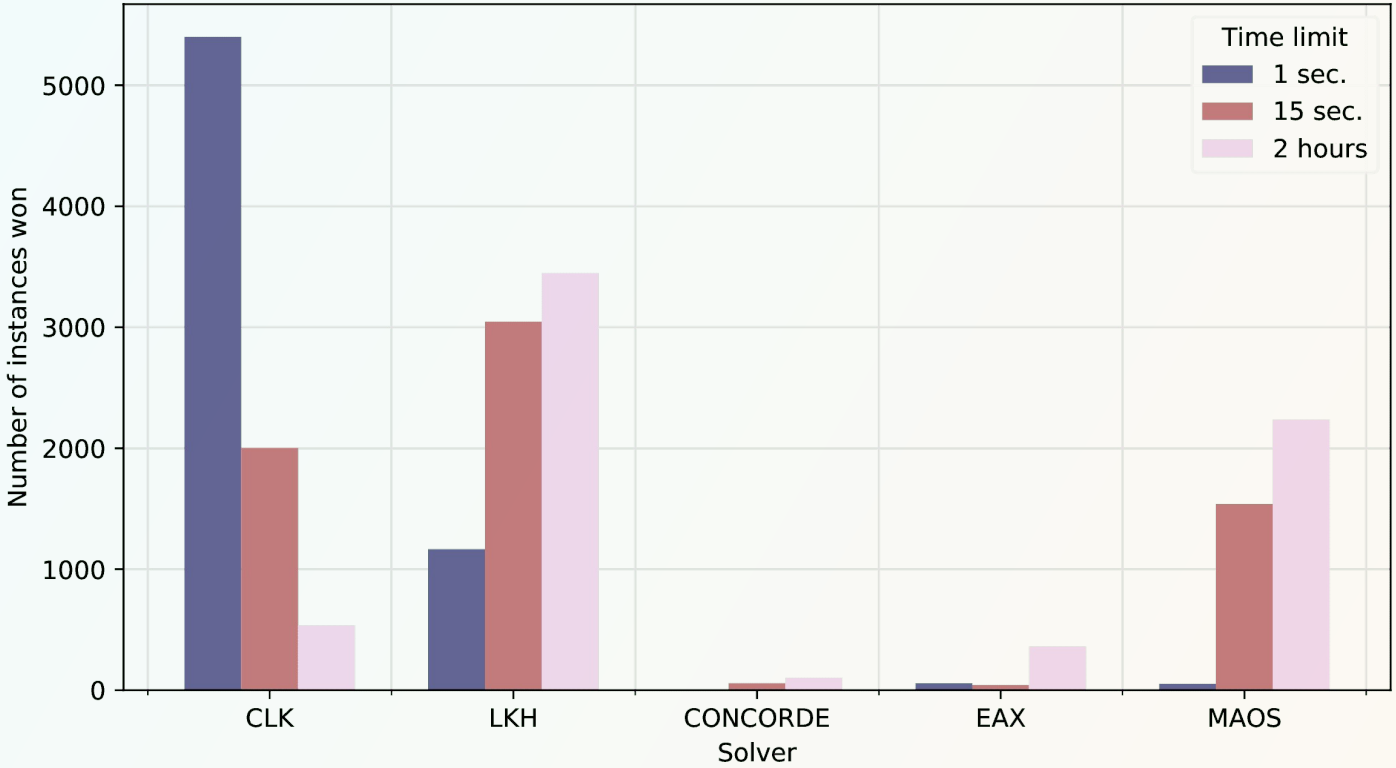
Output representation



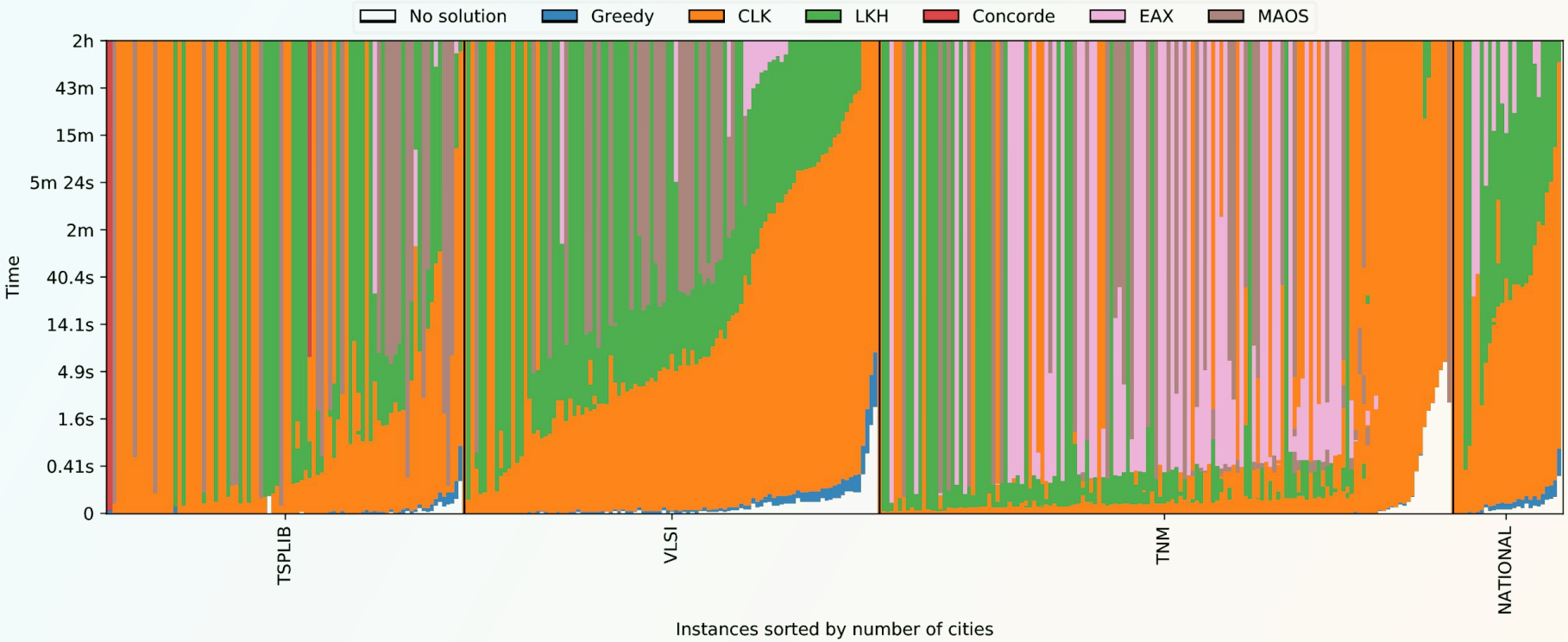
There are some problems in representing the neural network output with these values directly:

- ➔ Infinite value when there is no solution.
- ➔ Different instances will have different orders of magnitude in their costs.
- ➔ Minimal differences when close to the optimum.
- ➔ No tie-breaker information

Solver-Instance Execution

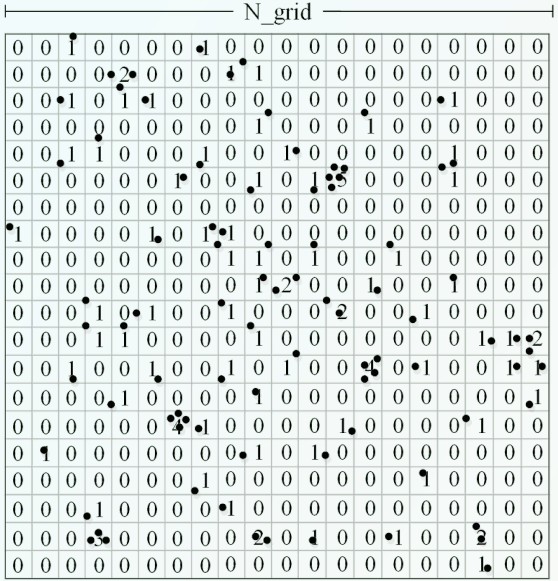


Solver-Instance Execution (Test data)



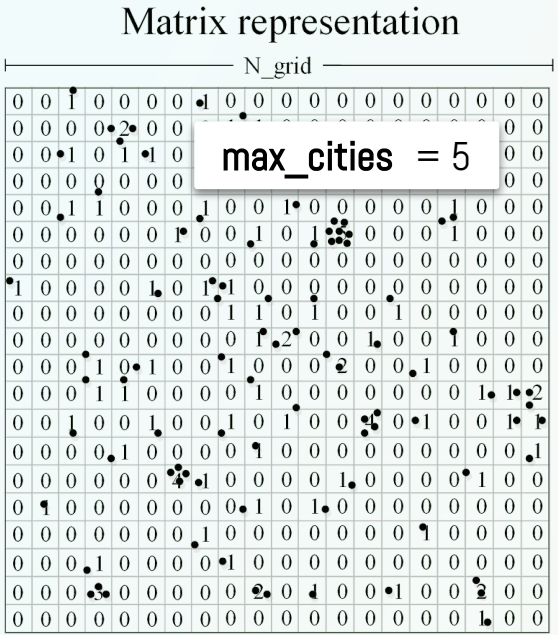
Parameters

Matrix representation



- **N_grid** : square matrix order.
- **max_cities** : maximum cell value. In order to normalize the matrix, the maximum value for the cells must be known.

Parameters



➔ **N_grid** : square matrix order.

➔ **max_cities** : maximum cell value. In order to normalize the matrix, the maximum value for the cells must be known.

Parameters

Matrix representation

N_grid

0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0											0	0
0	0	1	0	1	1	0											0	0
0	0	0	0	0	0	0											0	0
0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	1	0	1	5	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	2	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	0	1	0	0	0	2	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	2
0	0	1	0	0	1	0	0	1	0	1	0	0	4	0	1	0	0	1	1
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	4	1	0	0	0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	3	0	0	0	0	0	2	0	1	0	0	1	0	0	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

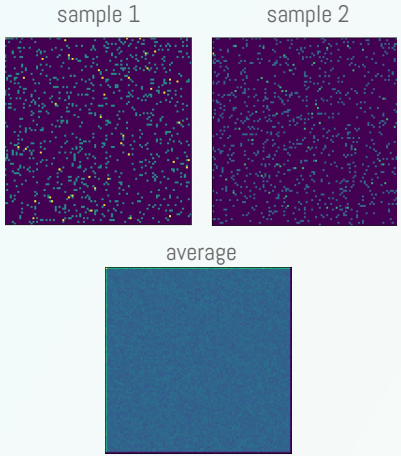
max_cities = 5

- **N_grid** : square matrix order.
- **max_cities** : maximum cell value. In order to normalize the matrix, the maximum value for the cells must be known.

Generated TSP Instances (Train)

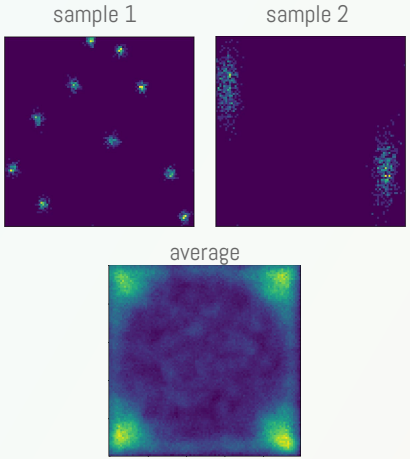
RUE [11]

3150 instances.
n random points in a square, $n \in [1, 2000]$.



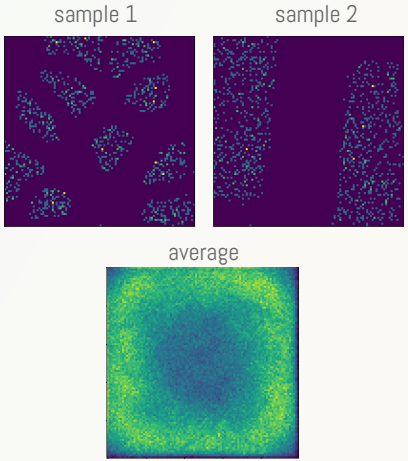
NETGEN [2]

600 instances.
Random clustered points.



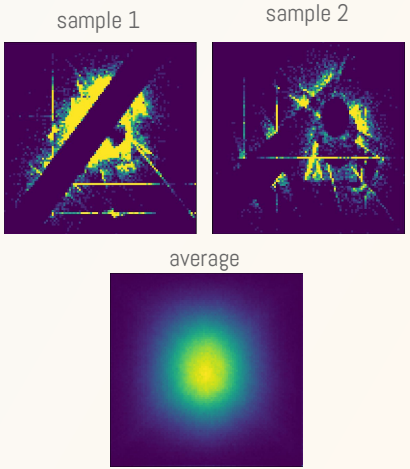
NETGENM [2]

600 instances.
Combine characteristics of RUE and NETGEN.



TSPGEN [12]

1981 instances.
Applies multiple mutation operators to a RUE instance.

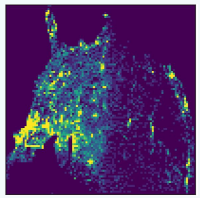


Public TSP Instances (Test)

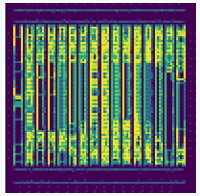
TSPLIB [8]

88 instances.
Different sources and types.

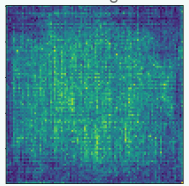
sample 1



sample 2



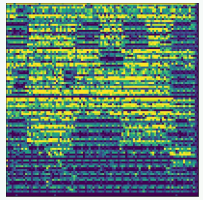
average



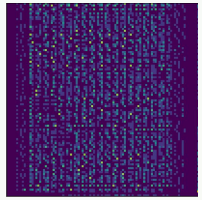
VLSI [9]

102 instances.
Very large-scale integration (VLSI) of integrated circuits.

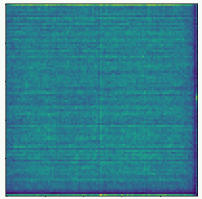
sample 1



sample 2



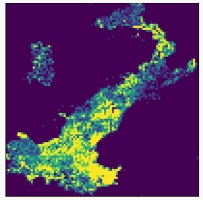
average



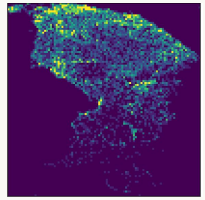
NATIONAL [9]

27 instances.
Subsets of the instances in World TSP grouped by country

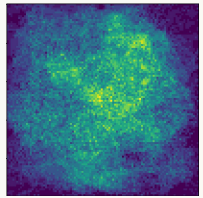
sample 1



sample 2



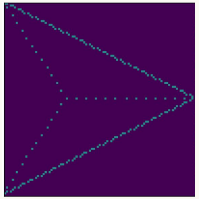
average



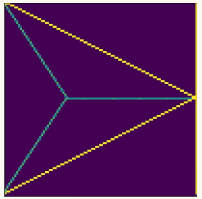
TNM [10]

141 instances.
Tetrahedron instance generator.

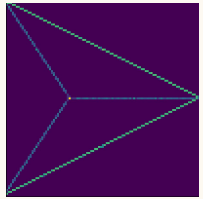
sample 1



sample 2



average



Solvers

Concorde

CLK | LKH | EAX | MAOS

Concorde (Applegate, 2003).

State-of-the-art in exact solvers for TSP. It uses the Branch & Bound search strategy, as well as plane cuts (Branch & Cut) to reduce the search space. A good initial solution is found using CLK.

Algorithm : Branch & Bound
Exact/Heuristic : Exact
Language : C

Solvers

CLK

LKH | EAX | MAOS | Concorde

Chained Lin Kernighan (Applegate, 2003).

Is a form of iterative local search. Within an iteration, it finds a local optima with an imprecise quality because it uses a k-opt neighborhood.

Algorithm : Iterated Local Search
Exact/Heuristic : Heuristic
Language : C

Solvers

LKH

EAX | MAOS | Concorde | CLK

Lin Kernighan Helsgaun (2018).

Is an improvement over the Lin Kernighan algorithm, proposed by Helsgaun (2000). It has been shown that with these improvements the algorithm gets closer to the optimal value in some instances, however, this also increases its execution times.

Algorithm : Iterated Local Search
Exact/Heuristic : Heuristic
Language : C

Solvers

EAX

MAOS | Concorde | CLK | LKH

Edge Assembly Crossover (Nagata & Kobayashi, 2013).

This is a genetic algorithm based on Edge Assembly Crossover, and it is considered one of the most effective recombination operators for the TSP. The usage of EAX allows for improvement in aspects such as localization, achieving higher effectiveness through the usage of local search algorithms to construct good parent solutions and therefore better children solutions.

Algorithm : Genetic Algorithm
Exact/Heuristic : Heuristic
Language : C++

Solvers

MAOS

Concorde | CLK | LKH | EAX

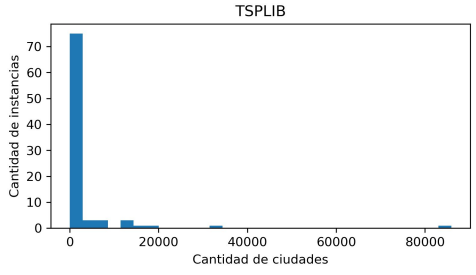
Multiagent Optimization System (Xie & Liu, 2018).

This method is based in self-organizing agents, that work with limited declarative knowledge and procedures under ecologic reasoning. Specifically, the agents explore in parallel, based on Socially Biased Individual Learning (SBIL) and interact indirectly with other agents through public environmentally organized information.

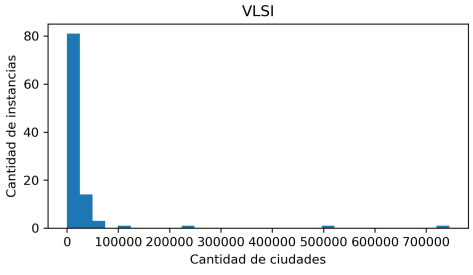
Algorithm : Multiagent Optimization System
Exact/Heuristic : Heuristic
Language : Java

Histogram test data

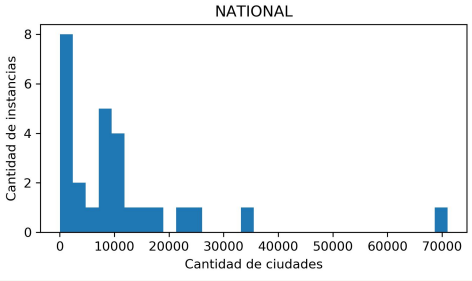
TSPLIB [8]



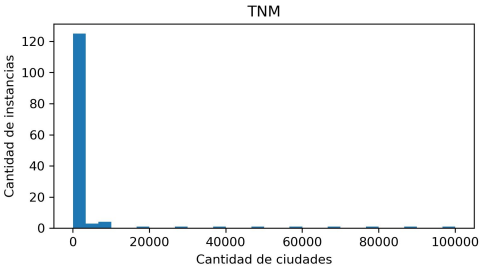
VLSI [9]



NATIONAL [9]



TNM [10]



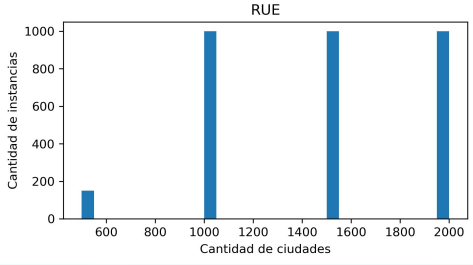
Histogram train data

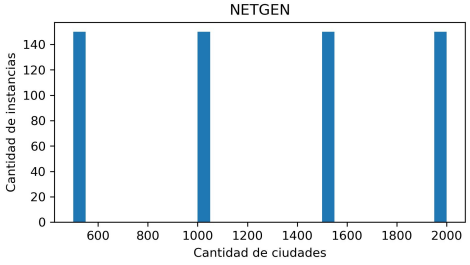
RUE 11

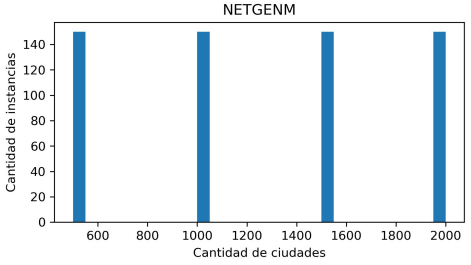
NETGEN 2

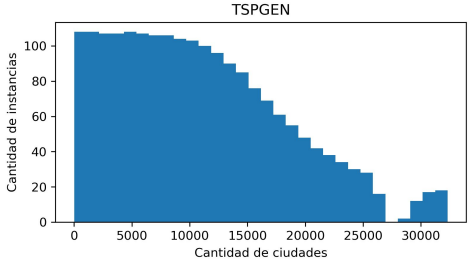
NETGENM 2

TSPGEN 12





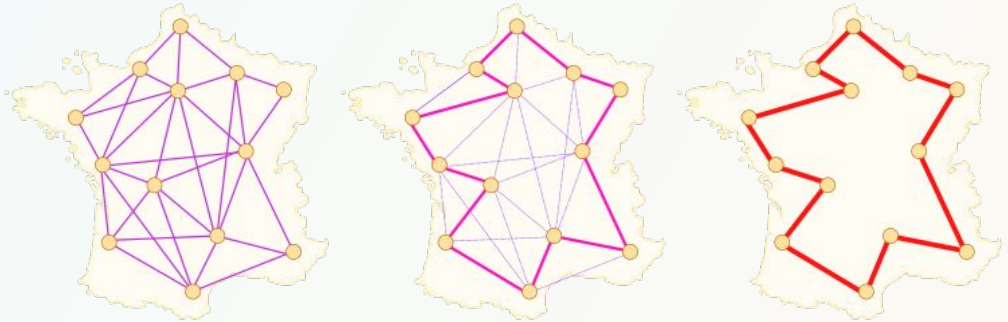




Traveling Salesman Problem

NP-hard problem. Given a collection of n cities connected by highways:

what is the **shortest route** that visits **every city** and returns to the starting place?



Euclidean TSP is a particular case of metric TSP, since distances in a plane obey the triangle inequality.

Staggered representation pseudocode

Algoritmo 1: Representación escalonada de una resolución para una instancia i

Input: $O_s(i,t), s \in \{1,2,3,4,5\}$

```
timeline ← [ ]
prev_positions ← [0, 0, 0, 0, 0]
for t = 1, ..., 500 do
    argsorted ← argsort([O_1(i,t), O_2(i,t), O_3(i,t), O_4(i,t), O_5(i,t)])
    curr_positions ← prev_positions
    for s = 4, ..., 0 do
        if argsorted[s] ≠ prev_argsorted[s] then
            if argsorted[s] = prev_argsorted[s − 1]
            and argsorted[s − 1] = prev_argsorted[s] then
                Swap positions
                curr_positions[argsorted[s]] ← curr_positions[argsorted[s − 1]]
                curr_positions[argsorted[s − 1]] ← curr_positions[argsorted[s]]
            else
                Find the index of change and increase its position and the upper
                positions by 1
                new_index ← argsorted.index(prev_argsorted[s])
                for ix = 0, ..., new_index do
                    curr_positions[prev_argsorted[ix]] += 1
                end
            end
        end
    end
    timeline[t] ← curr_positions
    prev_positions ← curr_positions
    prev_argsorted ← argsorted
end
return timeline
```

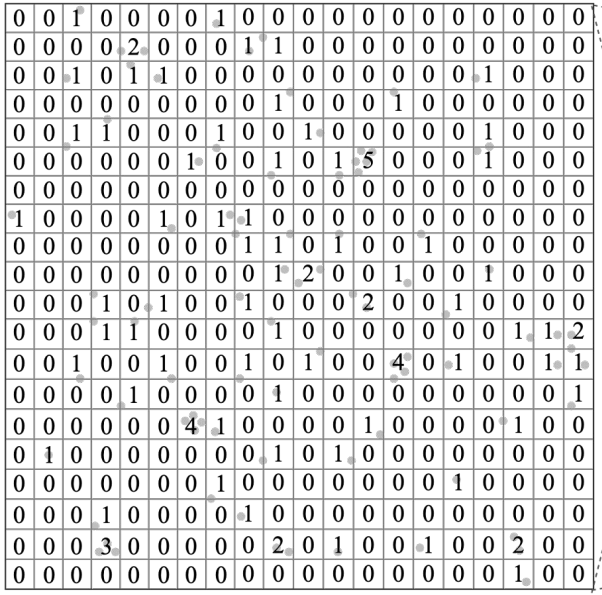
Improved prediction

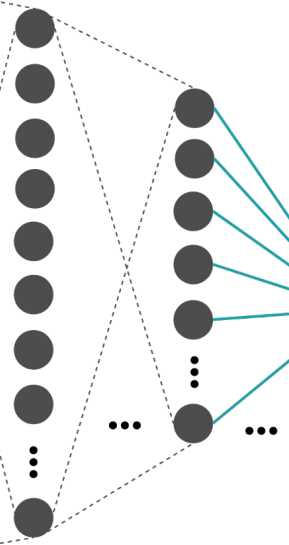
Representación matricial de la instancia

+

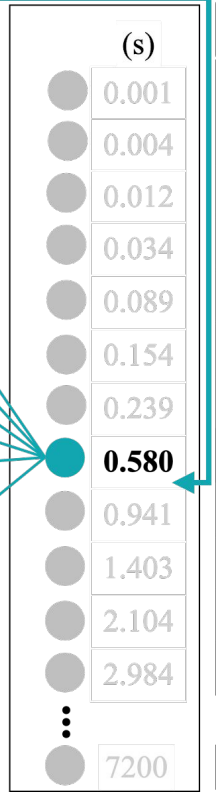
Tiempo

0.6 s





Capa Temporal



Optimalidad

EAX	LKH	CONC	CLK	MAOS
0.01	0.04	0	0.09	0
0.04	0.07	0.01	0.12	0.04
0.07	0.09	0.01	0.27	0.12
0.14	0.11	0.03	0.32	0.19
0.20	0.18	0.08	0.38	0.24
0.27	0.21	0.12	0.39	0.28
0.31	0.25	0.15	0.44	0.30
0.33	0.28	0.19	**0.48**	0.34
0.39	0.29	0.21	0.53	0.41
0.43	0.35	0.27	0.58	0.49
0.44	0.47	0.33	0.61	0.51
0.49	0.56	0.37	0.64	0.53
⋮	⋮	⋮	⋮	⋮
0.98	0.99	1	0.99	0.98