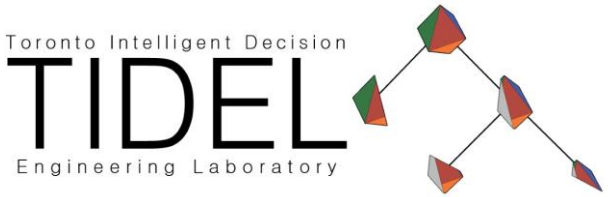
New Exact Methods for Solving Quadratic Traveling Salesman Problem

Jasper (Yuxiao) Chen[1], Anubhav Singh[1], **Ryo Kuroiwa**[2], and J. Christopher Beck[1]

[1]University of Toronto, Canada

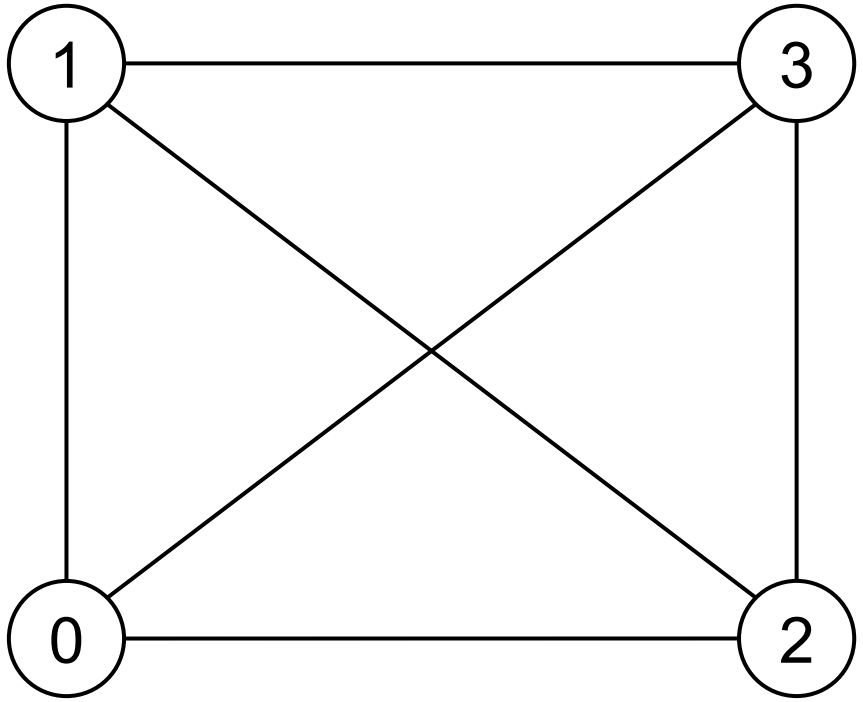
[2]National Institute of Informatics / SOKENDAI, Japan



Quadratic TSP (QTSP)

Traveling Salesperson Problem (TSP)

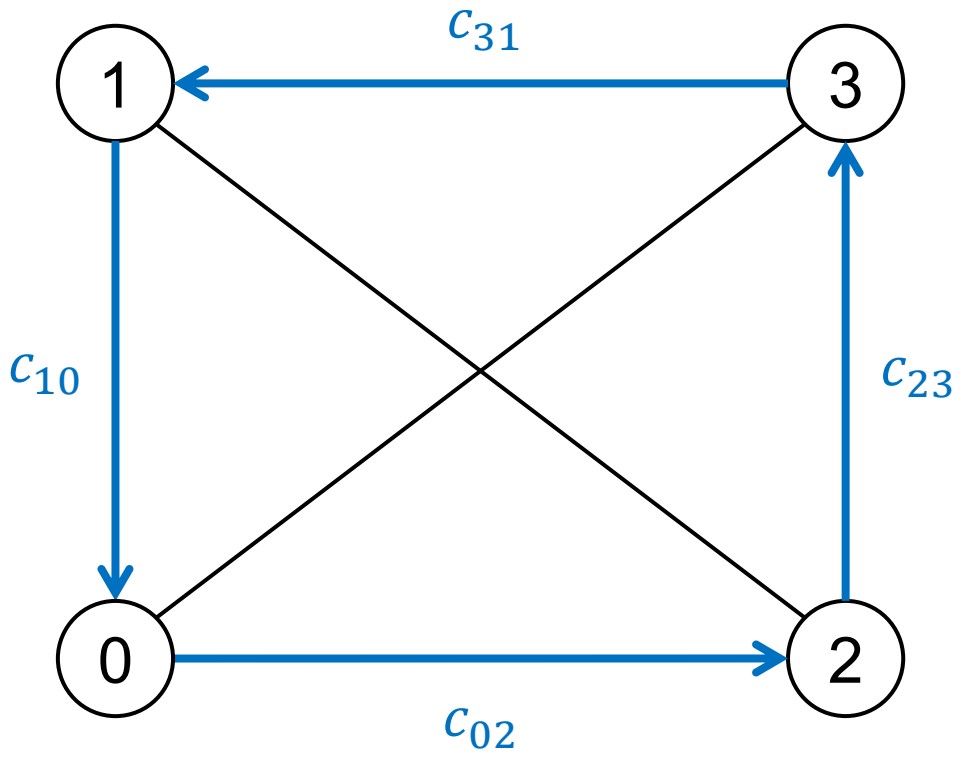
Minimize the tour cost to visit all locations



Traveling Salesperson Problem (TSP)

Minimize the tour cost to visit all locations

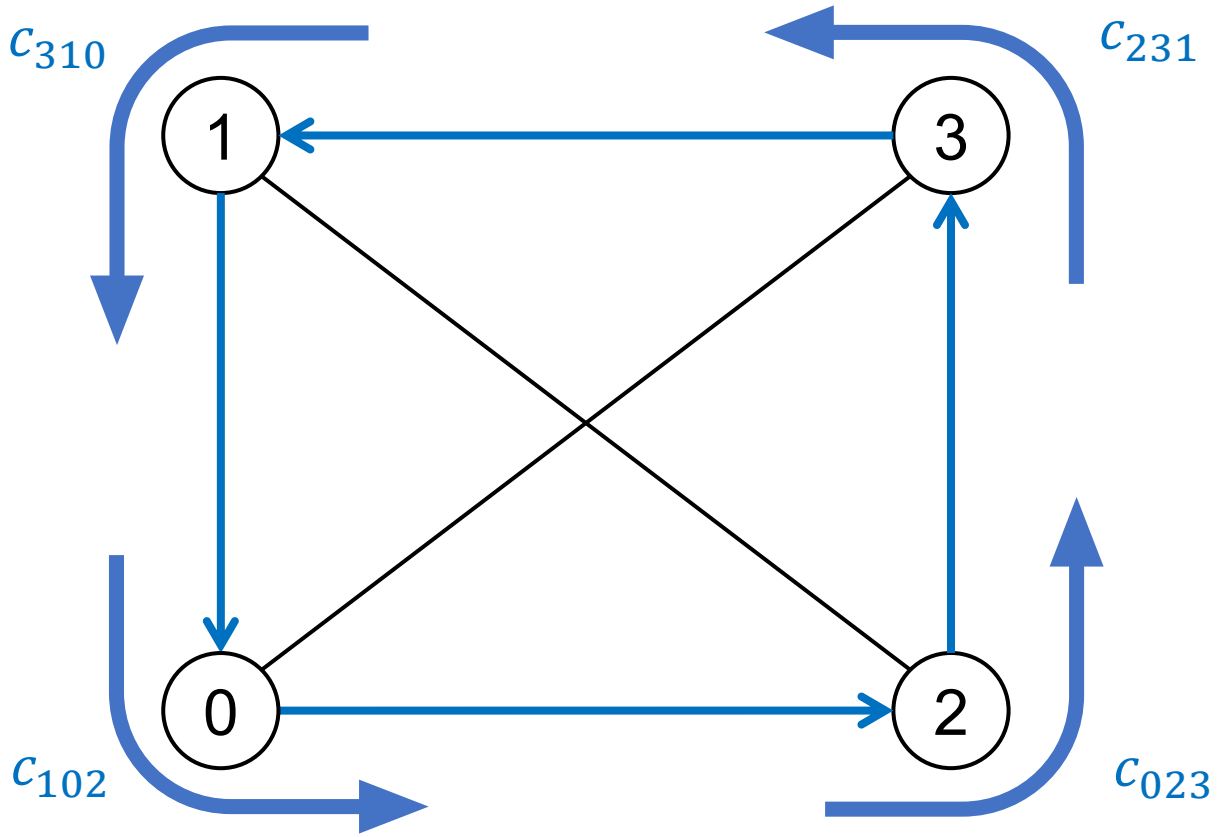
Cost: $c_{02} + c_{23} + c_{31} + c_{10}$



Quadratic TSP (QTSP)

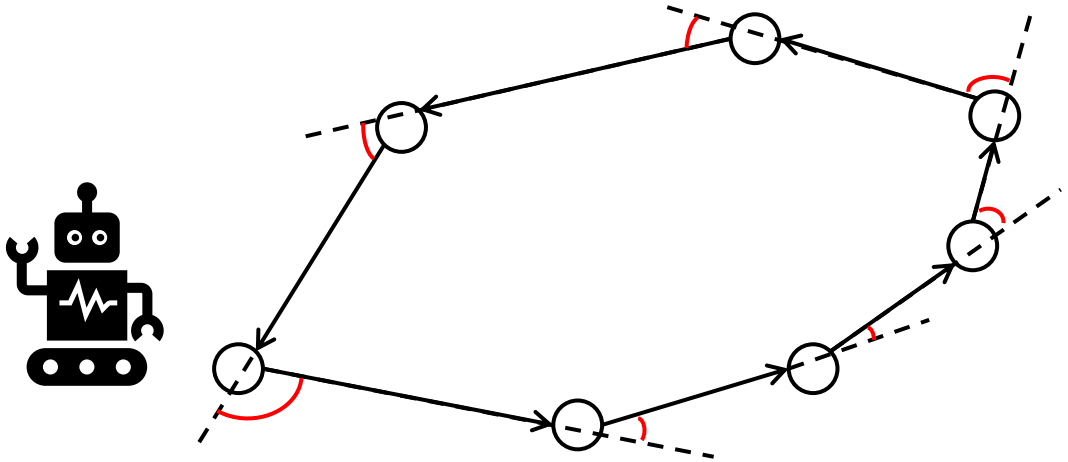
A cost is defined for each three consecutive visits

Cost: $c_{023} + c_{231} + c_{310} + c_{102}$



Motivations

Robotics: a smooth trajectory with small turning angles
[Aggrawal et al. 2000; Salva et al. 2008]



Bioinformatics: a permutation maximizing the log likelihood of a statistical model related to DNA sequences
[Jäger and Molitor 2008; Fischer et al. 2014]

Solving Approaches

Existing Approaches

Method	Exact	References
Linear Programming + Metaheuristics	No	Staněk et al. (2019)
Hybrid Genetic Algorithm	No	Pham et al. (ICAPS 2023)
Branch-and-Bound	Yes	Jäger and Molitor (2008)
Branch-and-Cut based on Integer Linear Programming	Yes	Jäger and Molitor (2008), Fischer et al. (2014), Oswin et al. (2017)
Translation to TSP	Yes	Fischer et al. (2014)
Dynamic Programming	Yes	Fischer et al. (2015)
Mixed-Integer Linear Programming	Yes	This work
Mixed-Integer Quadratic Programming	Yes	This work
Constraint Programming	Yes	This work
Domain-Independent Dynamic Programming	Yes	This work

Our Approaches

Method	Exact	References
Linear Programming + Metaheuristics	No	Staněk et al. (2019)
Hybrid Genetic Algorithm	No	Pham et al. (ICAPS 2023)
Branch-and-Bound	Yes	Jäger and Molitor (2008)
Branch-and-Cut based on Integer Linear Programming	Yes	Jäger and Molitor (2008), Fischer et al. (2014), Oswin et al. (2017)
Translation to TSP	Yes	Fischer et al. (2014)
Dynamic Programming	Yes	Fischer et al. (2015)
Mixed-Integer Linear Programming	Yes	This work
Mixed-Integer Quadratic Programming	Yes	This work
Constraint Programming	Yes	This work
Domain-Independent Dynamic Programming	Yes	This work

Mixed-Integer Linear Programming (MILP) Model

$N = \{0, \dots, n-1\}$: the set of locations

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} y_{ijk}$$

$$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij} \qquad \forall i \in N, j \in N \setminus \{i\}$$

Eliminate subtours (cycles consist of a strict subset)

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}$$

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}, k \in N \setminus \{i,j\}$$

Mixed-Integer Linear Programming (MILP) Model

$N = \{0, \dots, n-1\}$: the set of locations

$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} y_{ijk}$ The objective function

$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1$ $\forall i \in N$

$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij}$ $\forall i \in N, j \in N \setminus \{i\}$

Eliminate subtours (cycles consist of a strict subset)

$x_{ij} \in \{0,1\}$ $\forall i \in N, j \in N \setminus \{i\}$

$y_{ijk} \in \{0,1\}$ $\forall i \in N, j \in N \setminus \{i\}, k \in N \setminus \{i,j\}$

Visit i, j, and k consecutively or not

Mixed-Integer Linear Programming (MILP) Model

$N = \{0, \dots, n-1\}$: the set of locations

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} y_{ijk}$$

$$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij} \qquad \forall i \in N, j \in N \setminus \{i\}$$ Link x and y

Eliminate subtours (cycles consist of a strict subset)

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}$$ Visit j from i

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}, k \in N \setminus \{i,j\}$$

Visit i, j, and k consecutively or not

Mixed-Integer Linear Programming (MILP) Model

$N = \{0, \dots, n-1\}$: the set of locations

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk} y_{ijk}$$

$$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1 \qquad \forall i \in N$$

Each location is visited once

$$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij} \qquad \forall i \in N, j \in N \setminus \{i\}$$

Eliminate subtours (cycles consist of a strict subset)

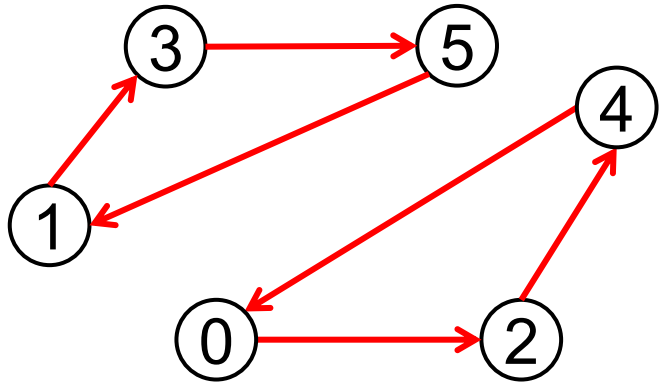
$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}$$

Visit j from i

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}, k \in N \setminus \{i,j\}$$

Mixed-Integer Linear Programming (MILP) Model

$N = \{0, \dots, n-1\}$: the set of locations



$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i.j\}} c_{ijk} y_{ijk}$$

$$\sum_{j \in N \setminus \{i\}} x_{ij} = \sum_{j \in N \setminus \{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$x_{ij} = \sum_{k \in N \setminus \{i,j\}} y_{ijk} = \sum_{k \in N \setminus \{i,j\}} y_{kij} \qquad \forall i \in N, j \in N \setminus \{i\}$$

Eliminate subtours (cycles consist of a strict subset)

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}$$

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N \setminus \{i\}, k \in N \setminus \{i,j\}$$

Mixed-Integer Linear Programming (MILP) Model

Eliminate subtours using continuous variables and constraints

Originally developed for TSP [Desrochers and Laporte 1991]

$$\min \sum_{i\in N}\sum_{j\in N\setminus\{i\}}\sum_{k\in N\setminus\{i,j\}} c_{ijk}y_{ijk}$$

$$\sum_{j\in N\setminus\{i\}} x_{ij} = \sum_{j\in N\setminus\{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$x_{ij} = \sum_{k\in N\setminus\{i,j\}} y_{ijk} = \sum_{k\in N\setminus\{i,j\}} y_{kij} \qquad \forall i \in N, j \in N\setminus\{i\}$$

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 \qquad \forall i \in N\setminus\{0\}, j \in N\setminus\{i,0\}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N\setminus\{i\}$$

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N\setminus\{i\}, k \in N\setminus\{i,j\}$$

$$1 \leq u_i \leq n-1 \qquad \forall i \in N\setminus\{0\}$$

Position of location i in the tour

$u_j \geq u_i + 1$ if $x_{ij} = 1$

Mixed-Integer Quadratic Programming (MIQP)

Replace y_{ijk} with $x_{ij}x_{jk}$ as quadratic terms are allowed

$$\min \sum_{i\in N}\sum_{j\in N\setminus\{i\}}\sum_{k\in N\setminus\{i,j\}} c_{ijk}\, x_{ij}x_{jk}$$

$$\sum_{j\in N\setminus\{i\}} x_{ij} = \sum_{j\in N\setminus\{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$u_i - u_j + (n-1)x_{ij} + (n-3)x_{ji} \leq n-2 \qquad \forall i \in N\setminus\{0\}, j \in N\setminus\{i,0\}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N\setminus\{i\}$$

$$1 \leq u_i \leq n-1 \qquad \forall i \in N\setminus\{0\}$$

Constraint Programming Model (CP-AD)

Realized by the all-different global constraint and element constraints

$$\min\ c_{x_{n-1}x_0x_1} + \sum_{i=0}^{n-3} c_{x_ix_{i+1}x_{i+2}} + c_{x_{n-2}x_{n-1}x_0}$$

$$\text{all\_different}(x_0, \dots, x_{n-1})$$ Each location is visited once

$$x_0 = 0$$ 0 is in the 0-th position wlog

$$x_i \in N \quad i = 0, \dots, n-1$$ Location x_i is in the i-th position

Constraint Programming Model (CP-CIR)

Realized by the circuit global constraint and logical constraints

$$\min \sum_{i \in N} \sum_{j \in N \setminus \{i\}} \sum_{k \in N \setminus \{i,j\}} c_{ijk}\, x_{ij} x_{jk}$$

$$\text{circuit}(z_0, \ldots, z_{n-1})$$ z forms a circuit

$$x_{ij} \leftrightarrow (z_i = j)\ \forall i \in N, j \in N \setminus \{i\}$$ Visit j from i

$$z_i \in N \setminus \{ i \}\ \ \forall i\ \in N$$ Location z_i is visited after location i

Domain-Independent Dynamic Programming (DIDP)

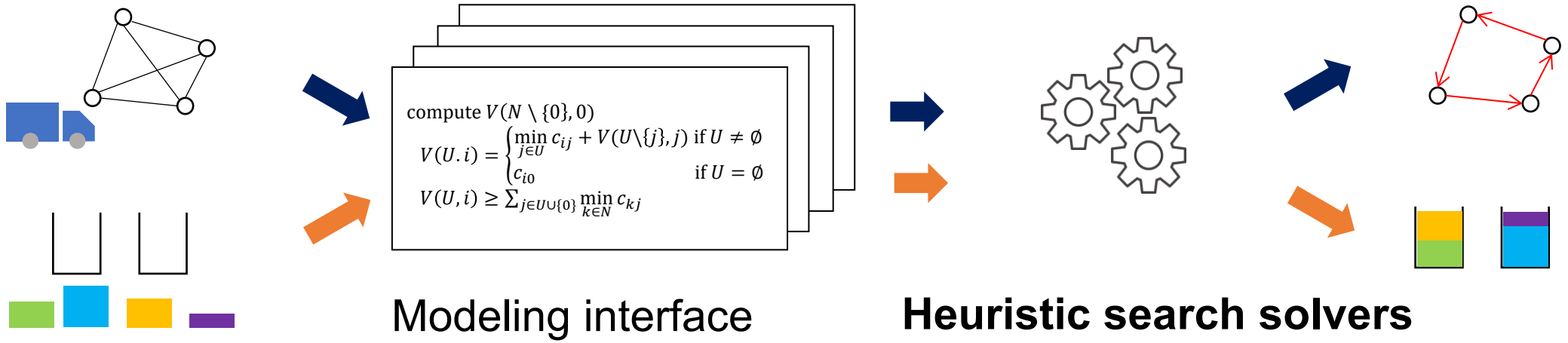
Model-based framework for dynamic programming (DP) based on state-based representations, inspired by PDDL planning

Combinatorial Optimization Problem

DP Model

General-Purpose DP Solver

Solution



Modeling interface
such as Python library

Heuristic search solvers
such as A* implemented in Rust

[Kuroiwa and Beck ICAPS 2023]

DP Model

Visit a location one by one, starting from location 0

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

DP Model

Visit a location one by one, starting from location 0

Bellman equation: V maps a state to its optimal solution cost

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

DP Model

Visit a location one by one, starting from location 0

Bellman equation: V maps a state to its optimal solution cost

compute $V(N \setminus \{0\}, 0, 0, 0)$ Initial state (no location is visited except for 0)

$$V(U,i,j,f) = \begin{cases} \min_{k \in U} V(U \setminus \{k\}, 0, k, k) & \text{if } j = 0 \\ \min_{k \in U} c_{ijk} + V(U \setminus \{k\}, j, k, f) & \text{if } j \neq 0 \wedge U \neq \emptyset \\ c_{jof} + c_{ij0} & \text{if } U = \emptyset \end{cases}$$

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

DP Model

Visit a location one by one, starting from location 0

Bellman equation: V maps a state to its optimal solution cost

$$\text{compute } V(N \setminus \{0\}, 0, 0, 0)$$

Initial state (no location is visited except for 0)

$$V(U,i,j,f) = \begin{cases} \min\limits_{k \in U} V(U \setminus \{k\}, 0, k, k) & \text{if } j = 0 \\ \min\limits_{k \in U} c_{ijk} + V(U \setminus \{k\}, j, k, f) & \text{if } j \neq 0 \wedge U \neq \emptyset \\ c_{j0f} + c_{ij0} & \text{if } U = \emptyset \end{cases}$$

Action (visit the first location k)

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

DP Model

Visit a location one by one, starting from location 0

Bellman equation: V maps a state to its optimal solution cost

$$\text{compute } V(N \setminus \{0\}, 0, 0, 0)$$

Initial state (no location is visited except for 0)

$$V(U, i, j, f) = \begin{cases} \min_{k \in U} V(U \setminus \{k\}, 0, k, k) & \text{if } j = 0 \\ \min_{k \in U} c_{ijk} + V(U \setminus \{k\}, j, k, f) & \text{if } j \neq 0 \wedge U \neq \emptyset \\ c_{j0f} + c_{ij0} & \text{if } U = \emptyset \end{cases}$$

Action (visit the first location k)

Action (visit a location k)

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

DP Model

Visit a location one by one, starting from location 0

Bellman equation: V maps a state to its optimal solution cost

$$\text{compute } V(N \setminus \{0\}, 0, 0, 0)$$

Initial state (no location is visited except for 0)

$$V(U,i,j,f) = \begin{cases} \min\limits_{k \in U} V(U \setminus \{k\}, 0, k, k) & \text{if } j = 0 \\ \min\limits_{k \in U} c_{ijk} + V(U \setminus \{k\}, j, k, f) & \text{if } j \neq 0 \wedge U \neq \emptyset \\ c_{j0f} + c_{ij0} & \text{if } U = \emptyset \end{cases}$$

Action (visit the first location k)

Action (visit a location k)

Goal (all locations are visited)

State (U, i, t, f):

- U: the set of unvisited locations
- i: the previous location
- j: the current location
- f: the first location visited after 0

Dual Bound

In DIDP, a **lower bound on the solution cost** can be explicitly modeled, and a solver can use it as an **admissible heuristic function**

Dual Bound (DIDP-1)

In DIDP, a **lower bound on the solution cost** can be explicitly modeled, and a solver can use it as an **admissible heuristic function**

Idea: visiting location k requires at least $\min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$

Dual Bound (DIDP-1)

In DIDP, a **lower bound on the solution cost** can be explicitly modeled, and a solver can use it as an **admissible heuristic function**

Idea: visiting location k requires at least $\min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$

$$V(U, i, j, f) \geq \sum_{k \in U \cup \{f, 0\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$$

Dual Bound (DIDP-1)

In DIDP, a **lower bound on the solution cost** can be explicitly modeled, and a solver can use it as an **admissible heuristic function**

Idea: visiting location k requires at least $\min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$

$$V(U, i, j, f) \geq \sum_{k \in U \cup \{f, 0\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$$

$O(n)$ for each state Precomputed for each k

Dual Bound (DIDP-1)

In DIDP, a **lower bound on the solution cost** can be explicitly modeled, and a solver can use it as an **admissible heuristic function**

Idea: visiting location k requires at least $\min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$

$$V(U, i, j, f) \geq \sum_{k \in U \cup \{f, 0\}} \min_{l \in N \setminus \{k\}, m \in N \setminus \{k,l\}} c_{lmk}$$

Two similar bounds defined with c_{lkm} and c_{klm} (the maximum is used)

Tighter Dual Bound (DIDP-2)

If location e was visited before the previous location, visiting k requires at least $\min_{l \in N \setminus \{k,e\}, m \in N \setminus \{k,l,e\}} c_{lmk}$

$$V(U,i,j,f) \geq \sum_{k \in U \cup \{f,0\}} \max_{e \in N \setminus (U \cup \{0,i,j\})} \min_{l \in N \setminus \{k,e\}, m \in N \setminus \{k,l,e\}} c_{lmk}$$

Two similar bounds defined with c_{lkm} and c_{klm} (the maximum is used)

Tighter Dual Bound (DIDP-2)

If location e was visited before the previous location, visiting k requires at least $\min_{l \in N \setminus \{k,e\}, m \in N \setminus \{k,l,e\}} c_{lmk}$

$$V(U,i,j,f) \geq \sum_{k \in U \cup \{f,0\}} \max_{e \in N \setminus (U \cup \{0,i,j\})} \min_{l \in N \setminus \{k,e\}, m \in N \setminus \{k,l,e\}} c_{lmk}$$

$O(n^2)$ for each state

Precomputed for each k and e

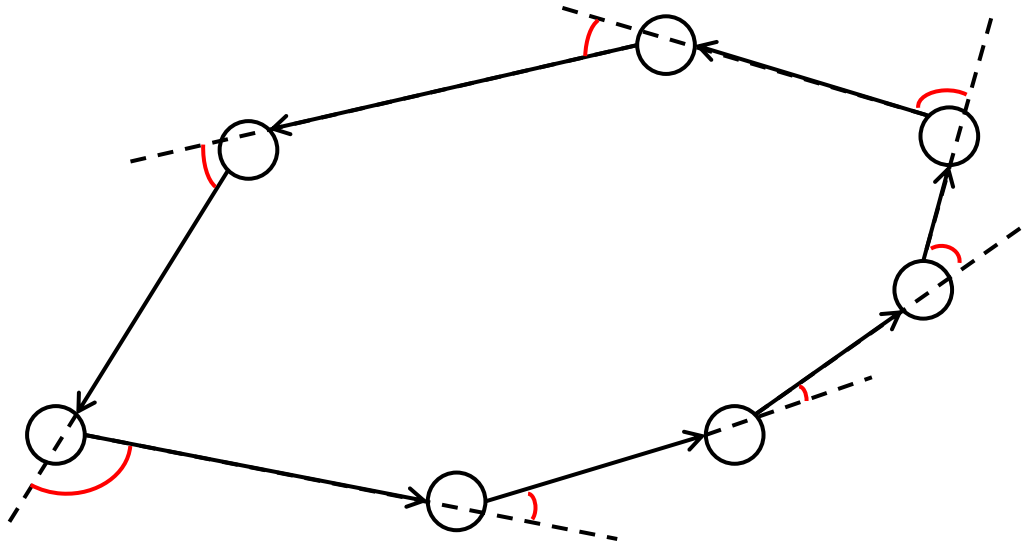
Two similar bounds defined with c_{lkm} and c_{klm} (the maximum is used)

Experimental Evaluation

Problem Instances

- **Angle-TSP** [Aggrawal et al. 2000]: minimize the sum of turning angles
- **AngleDistance-TSP** [Salva et al. 2008]: minimize the weighted sum of turning angles and distances

For each setting, 400 instances with 5-200 locations were generated by Staněk et al. (2019) + 40 instances with 250-400 locations by us



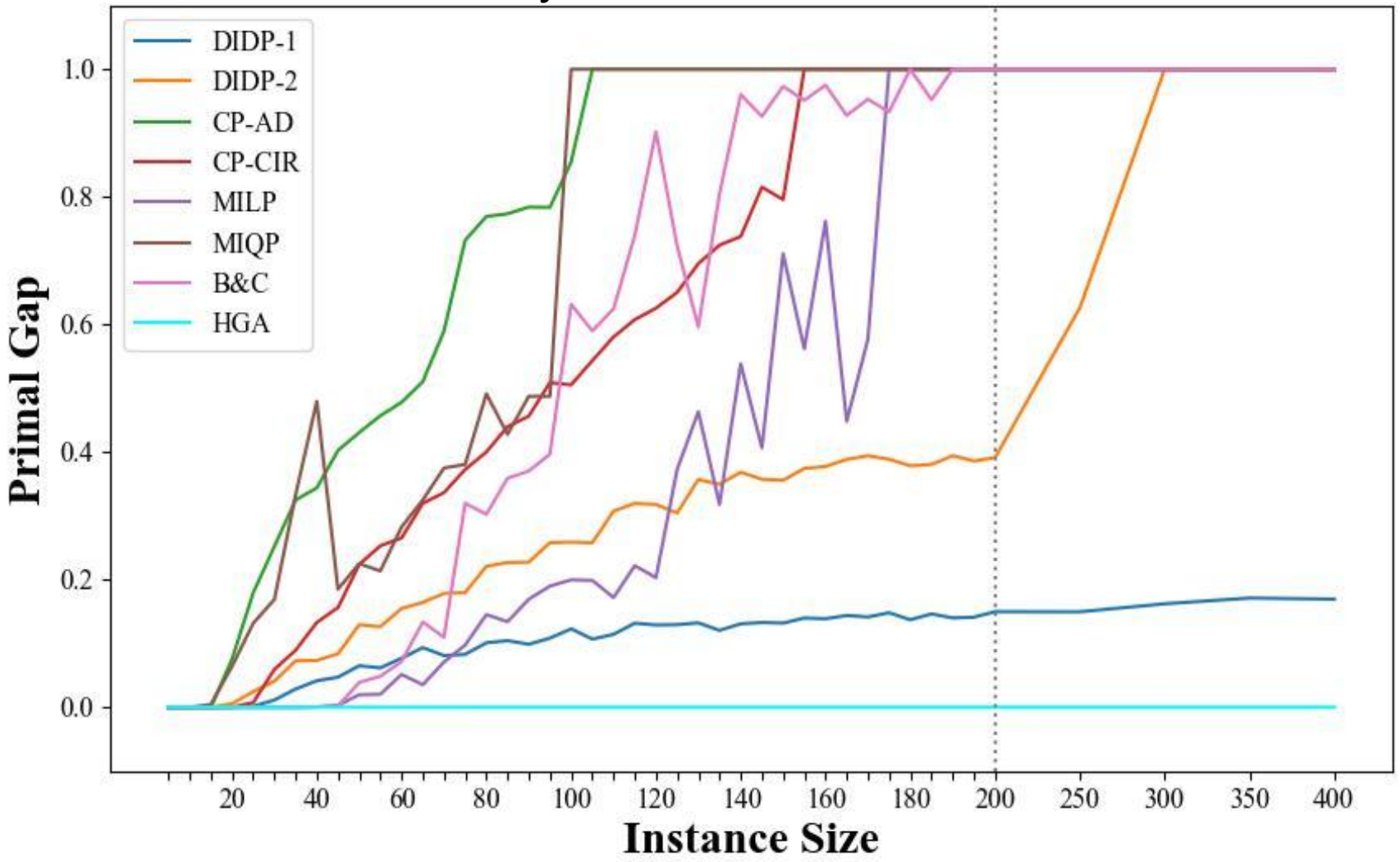
Evaluated Methods

Method	Solver	Exact	References
Hybrid Genetic Algorithm (HGA) (best in inexact methods)	Implemented by Pham et al. (2023)	No	Pham et al. (2023)
B&C based on ILP (best in existing exact methods)	Gurobi 11.0.3	Yes	Oswin et al. (2017)
MILP	Gurobi 11.0.3	Yes	This work
MIQP	Gurobi 11.0.3	Yes	This work
CP-AD, CP-CIR	CP Optimizer 22.1.1.0	Yes	This work
DIDP-1, DIDP-2	CABS in didp-rs 0.8.0	Yes	This work

30-min time and 8GB memory for up to 200 locations, 16GB for more

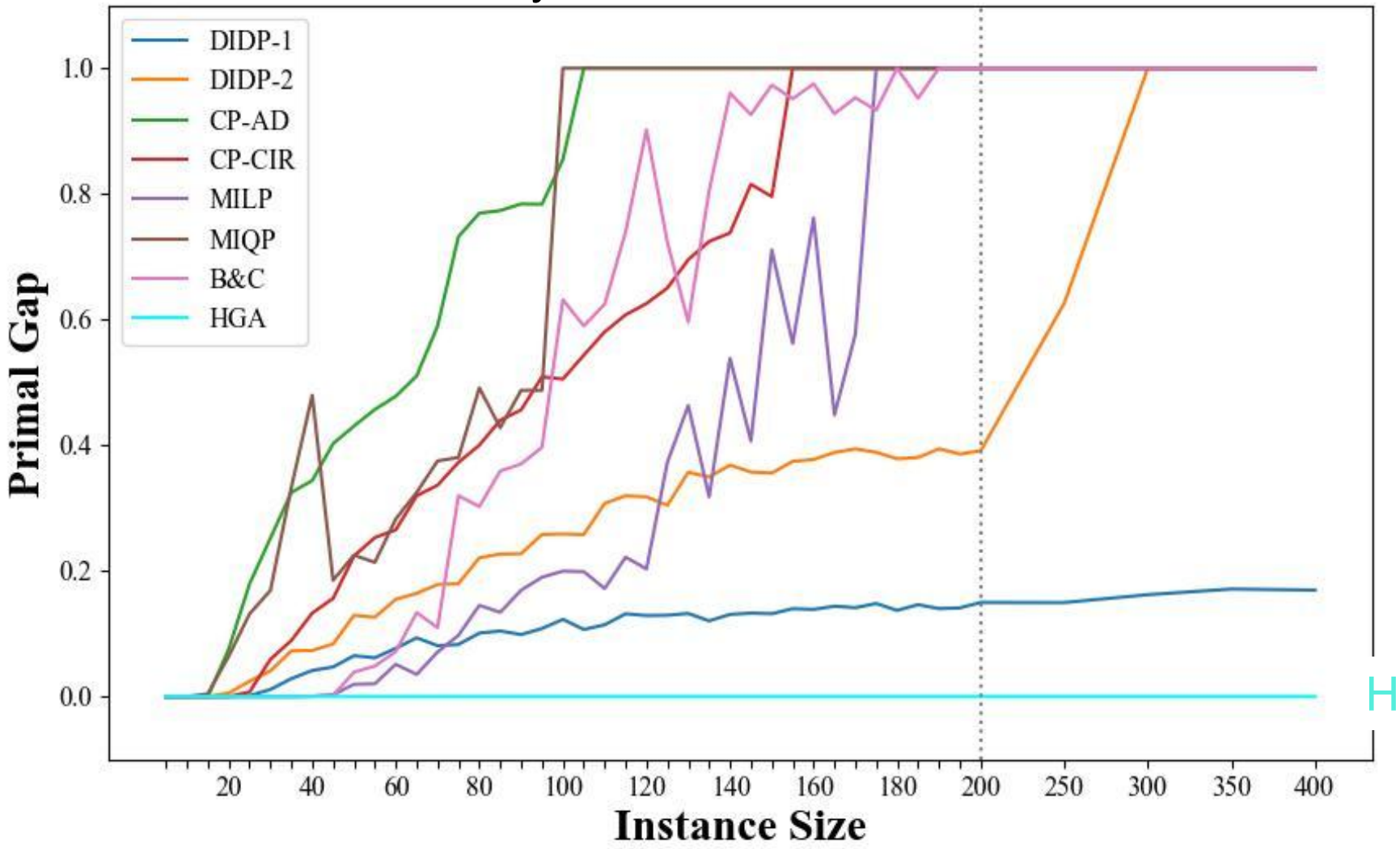
Average Primal Gap in Angle-TSP

Normalized solution cost found by a solver, 1: no solution, **smaller is better**



Average Primal Gap in Angle-TSP

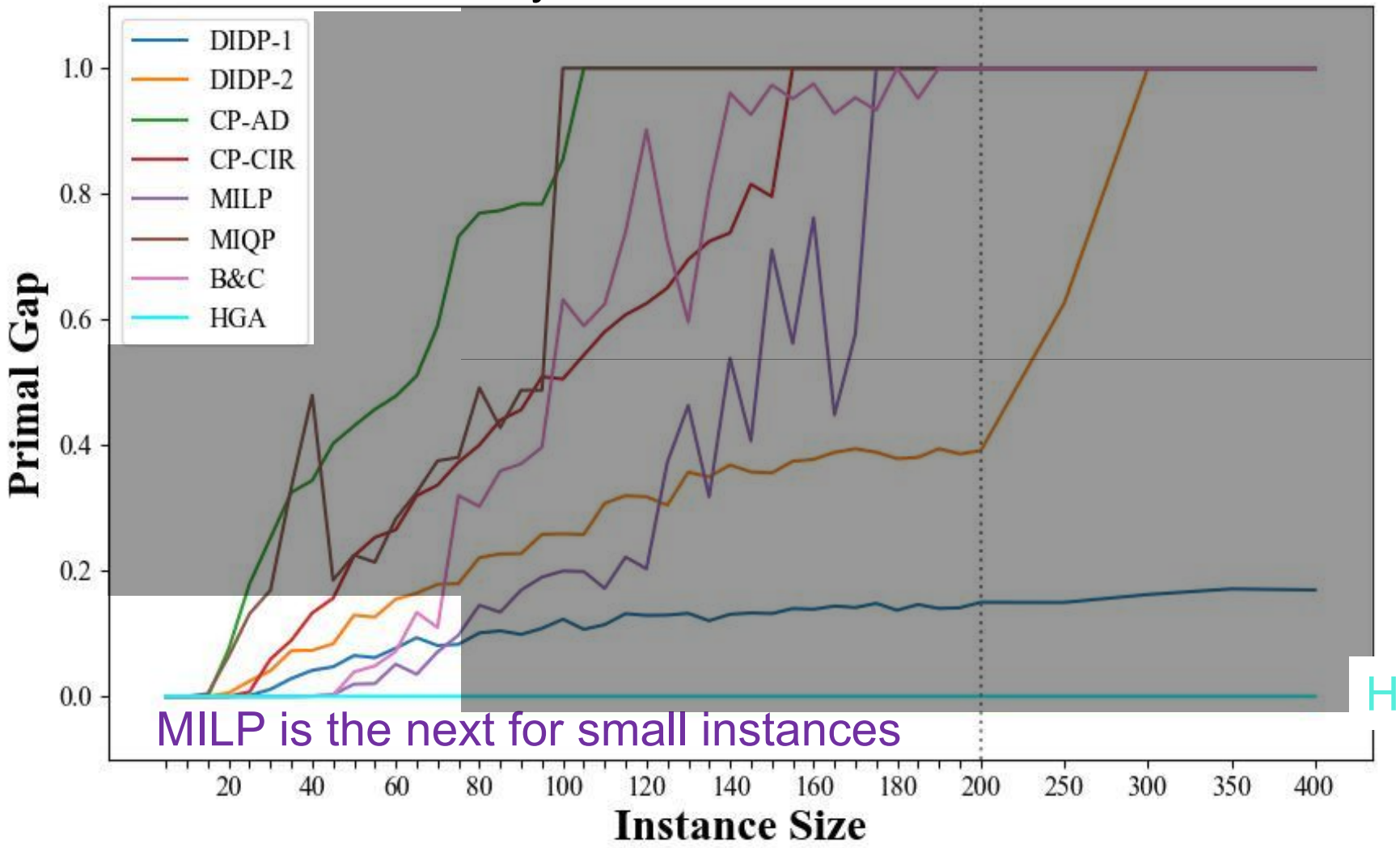
Normalized solution cost found by a solver, 1: no solution, **smaller is better**



HGA is the best

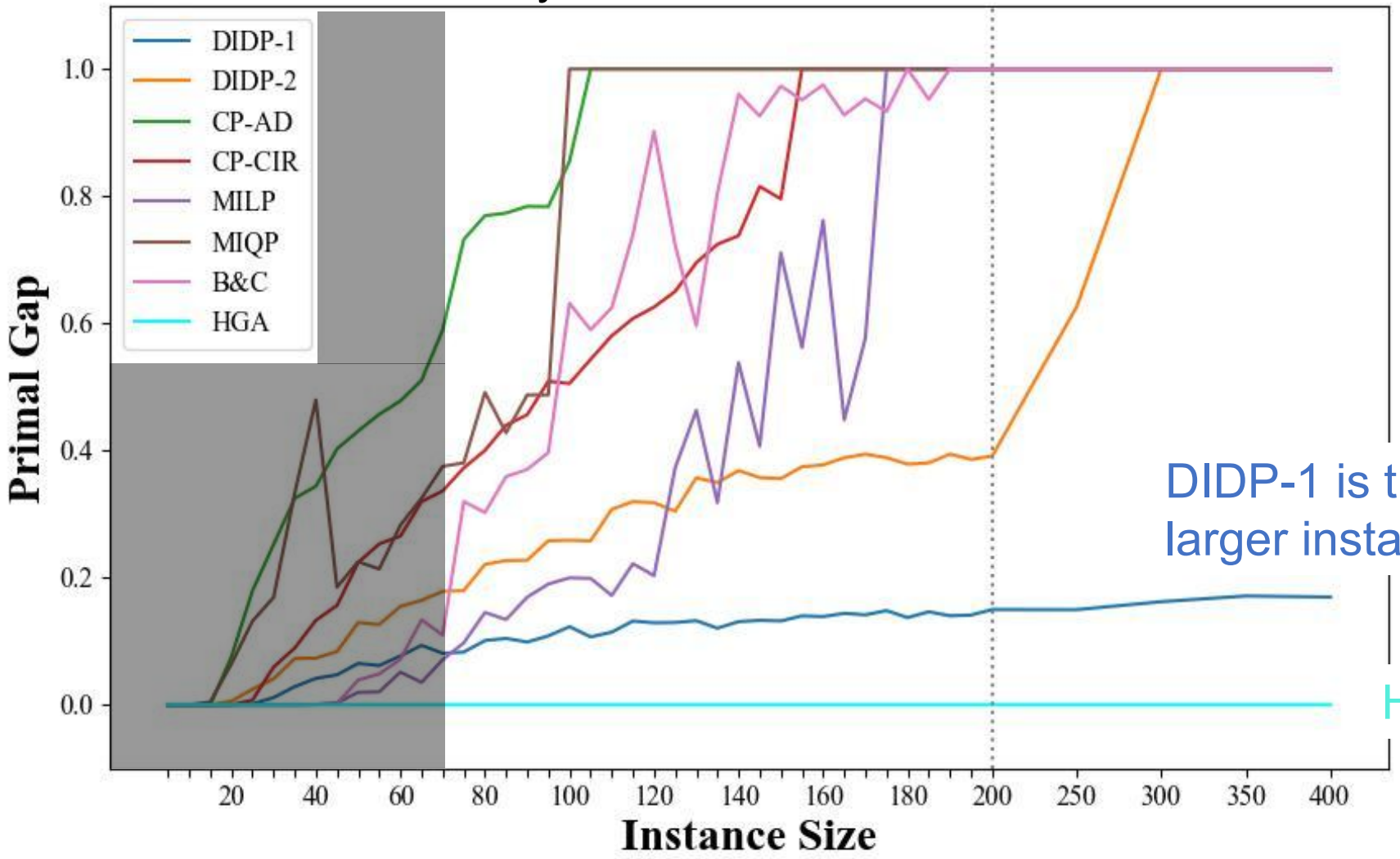
Average Primal Gap in Angle-TSP

Normalized solution cost found by a solver, 1: no solution, **smaller is better**



Average Primal Gap in Angle-TSP

Normalized solution cost found by a solver, 1: no solution, **smaller is better**

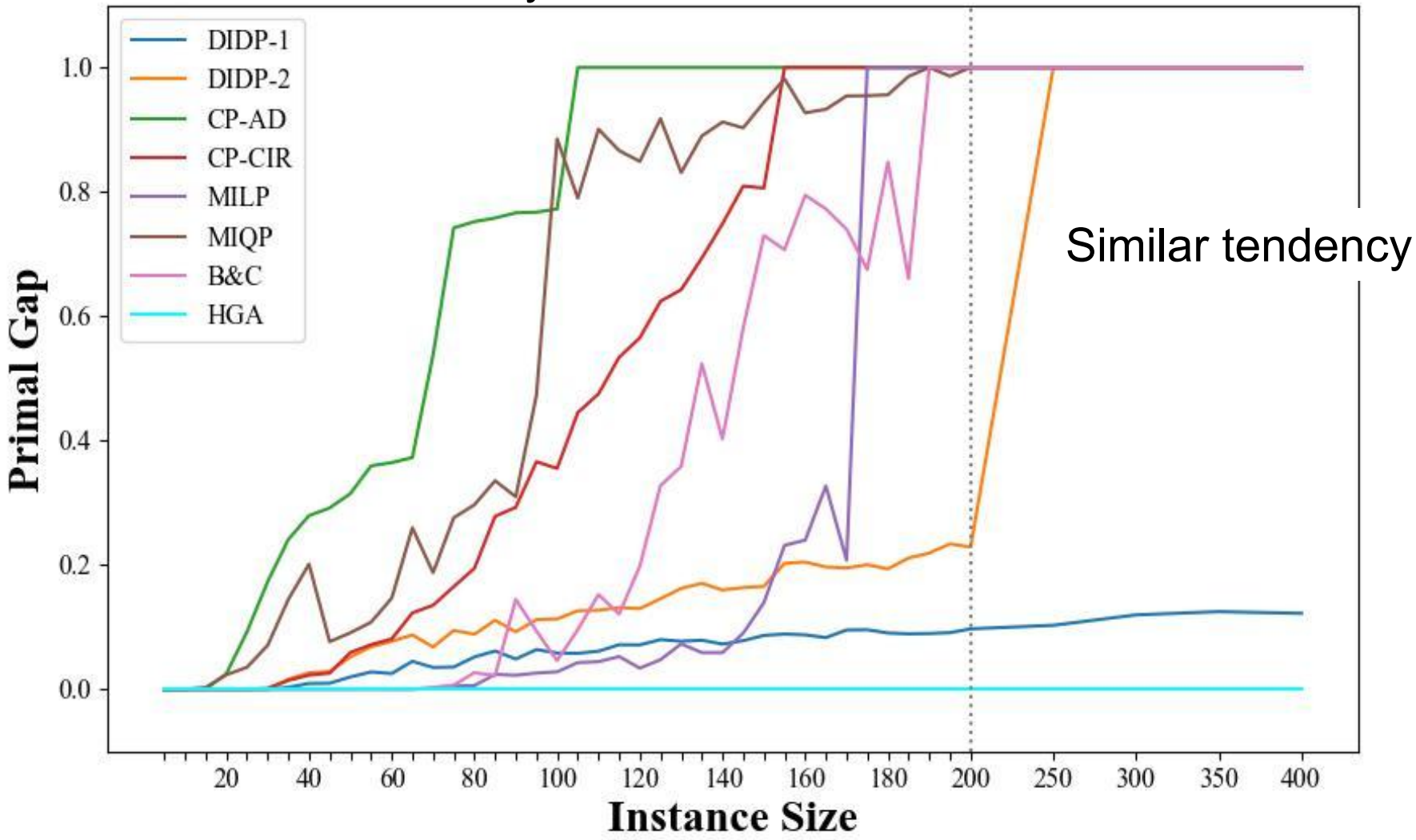


DIDP-1 is the next for larger instances

HGA is the best

Average Primal Gap in AngleDistance-TSP

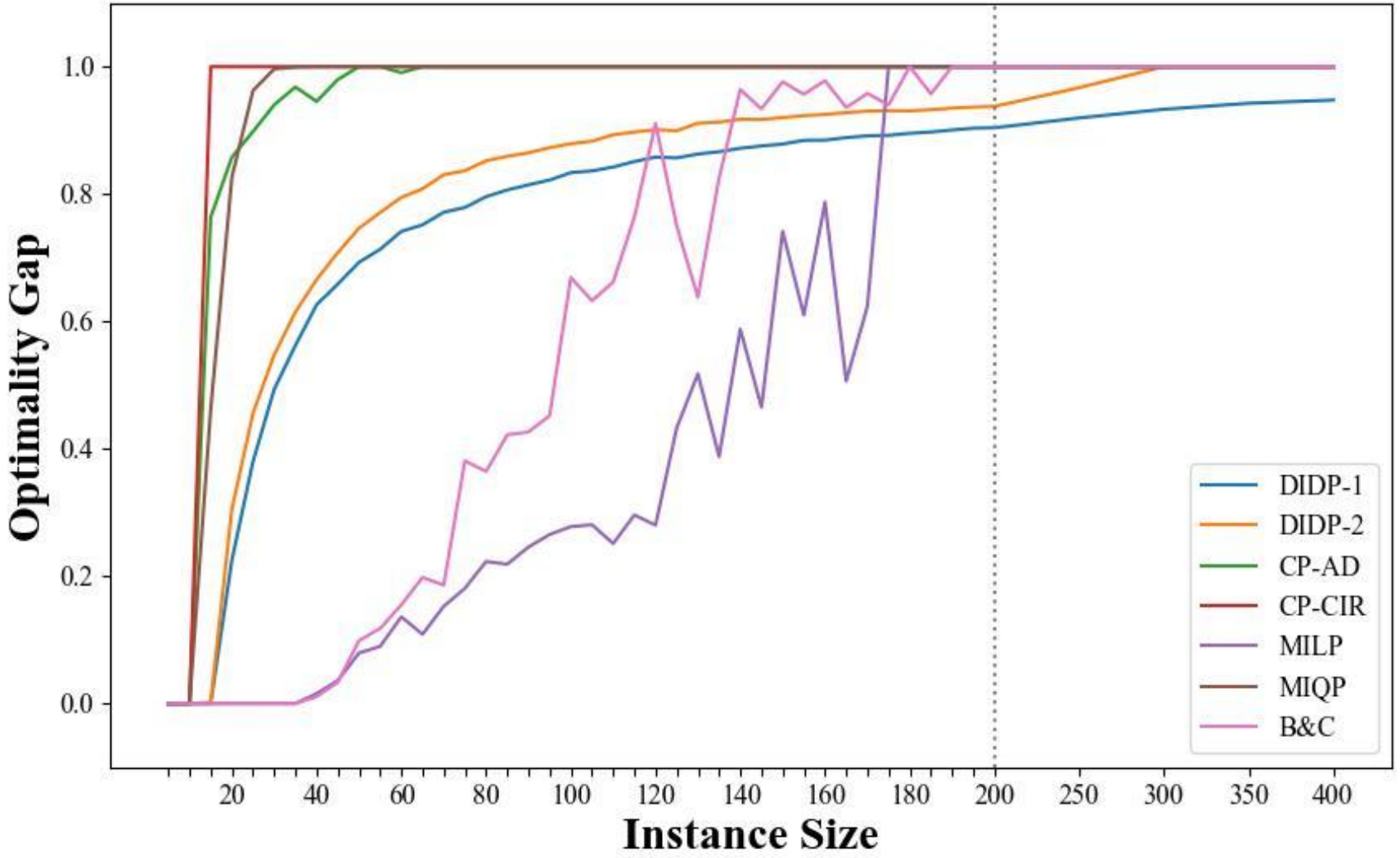
Normalized solution cost found by a solver, 1: no solution, **smaller is better**



Similar tendency

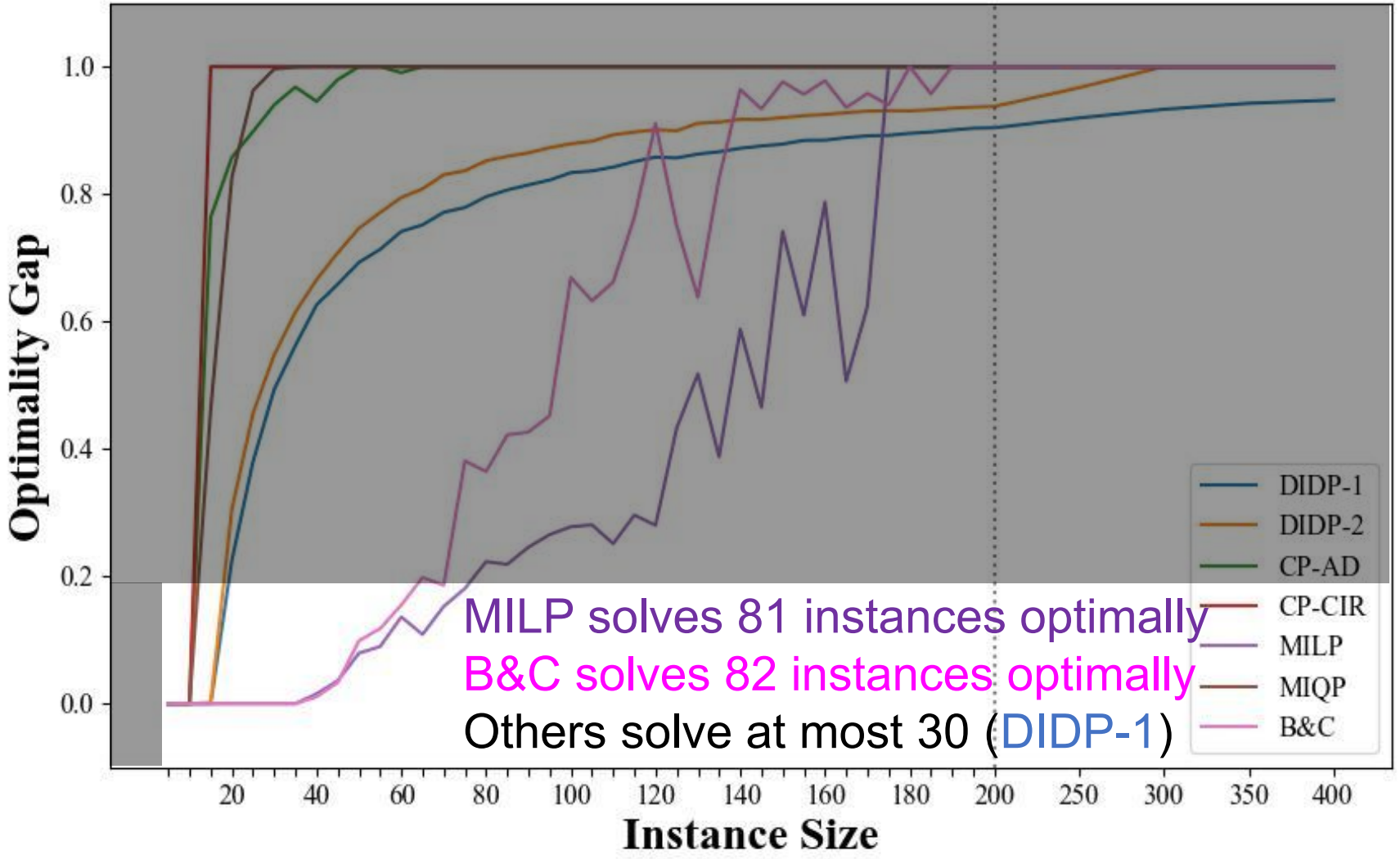
Average Optimality Gap in Angle-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



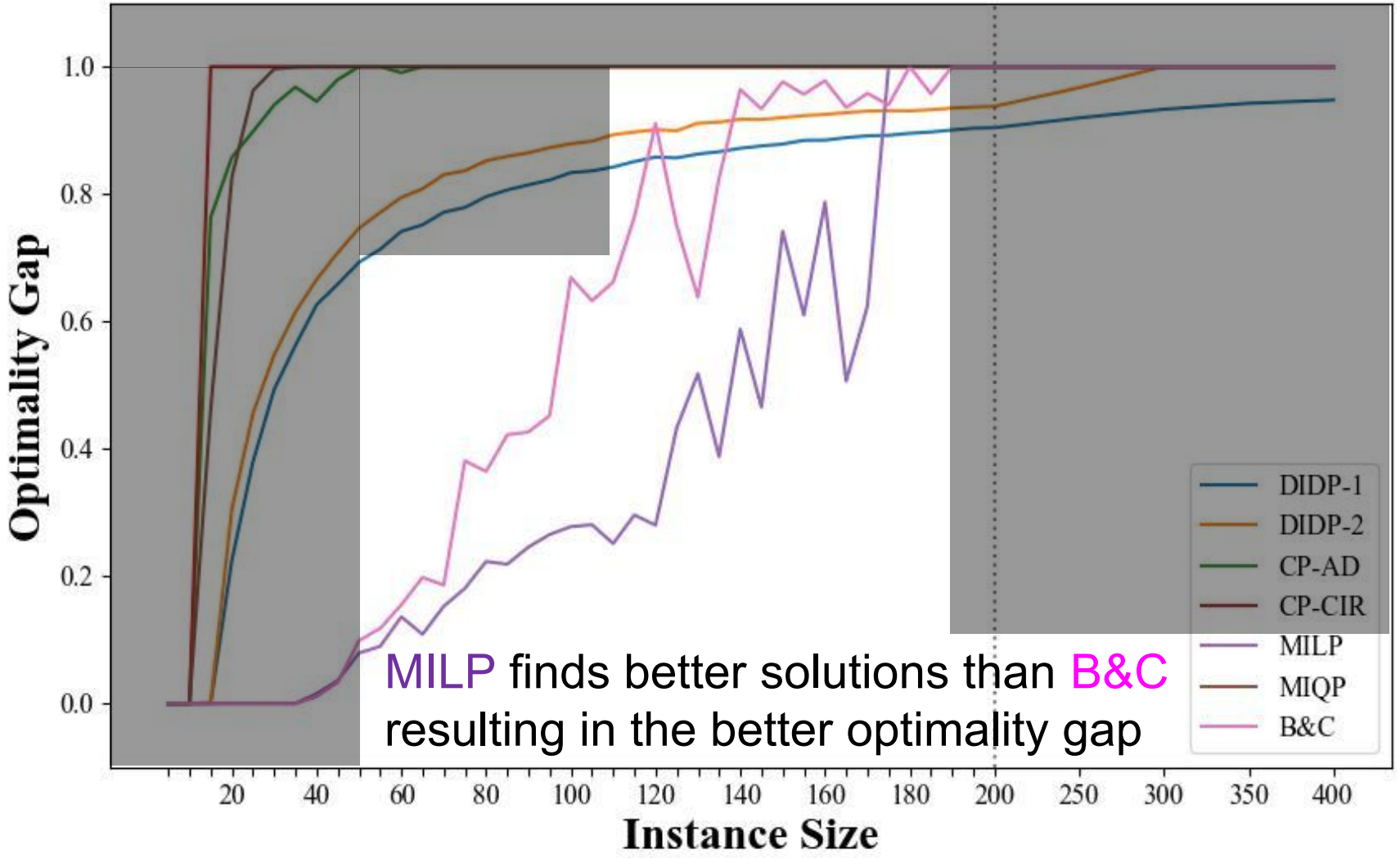
Average Optimality Gap in Angle-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



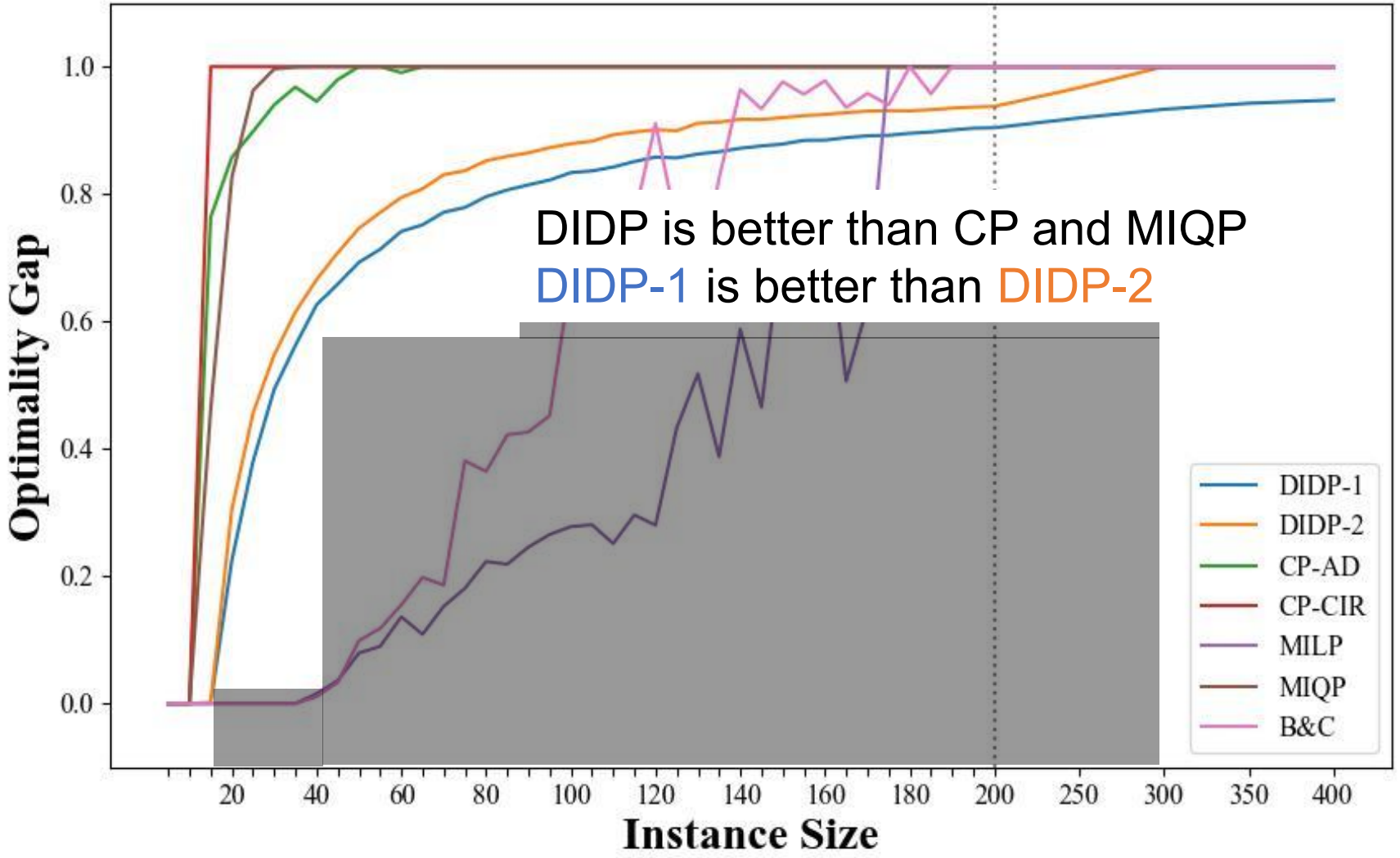
Average Optimality Gap in Angle-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



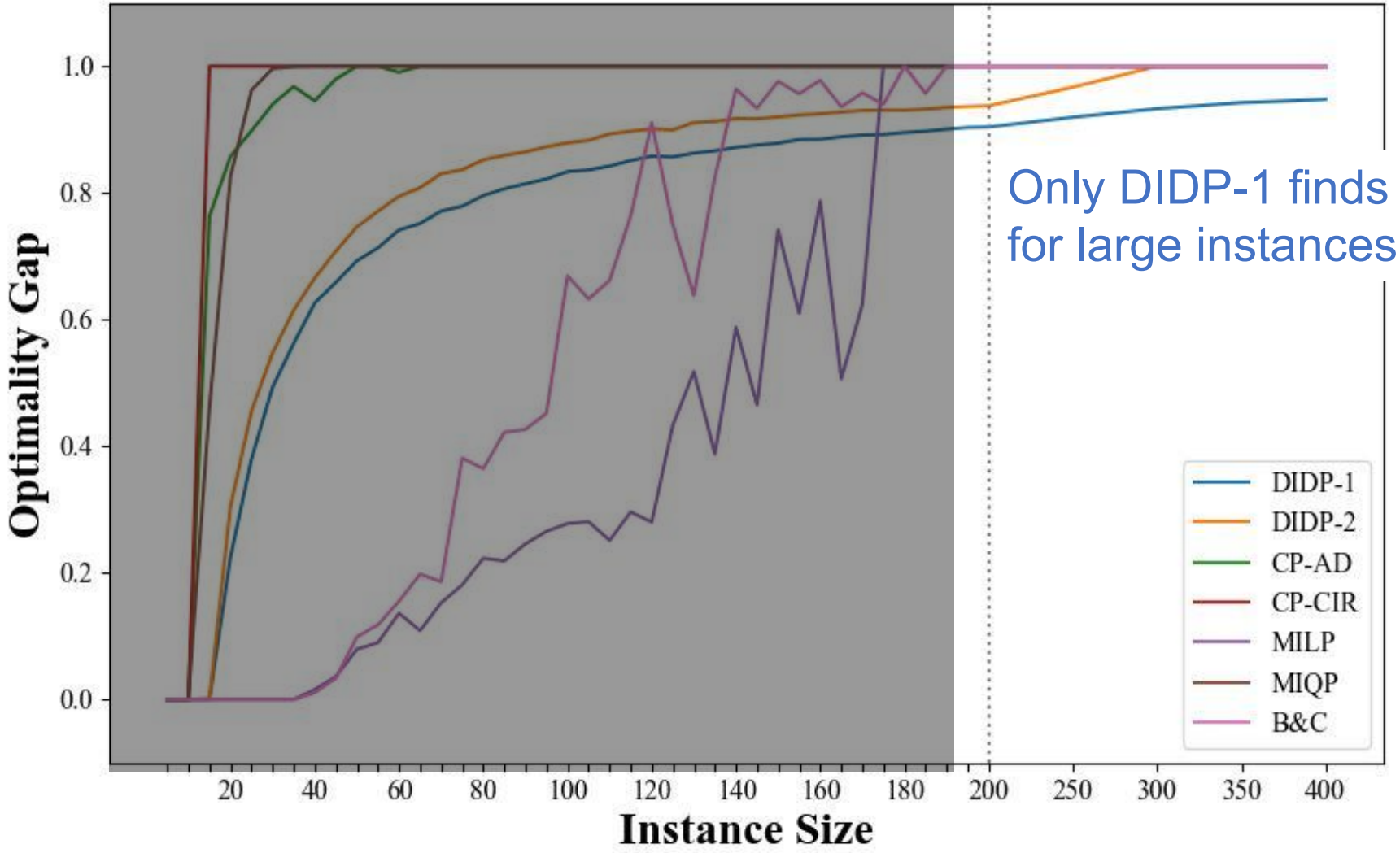
Average Optimality Gap in Angle-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



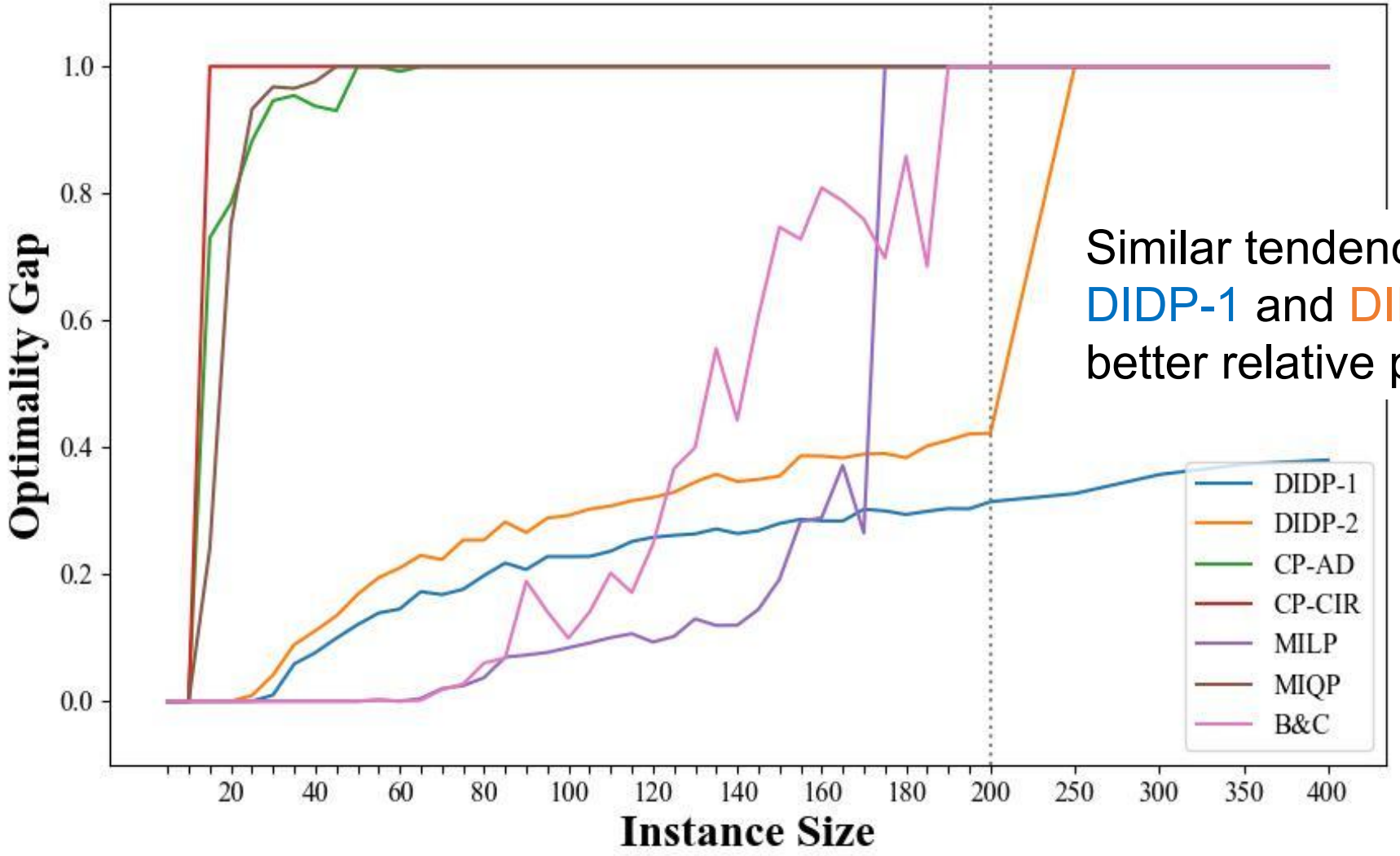
Average Optimality Gap in Angle-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



Average Optimality Gap in AngleDistance-TSP

Gap between upper and lower bounds, 0: optimal, 1: no solution, **smaller is better**



Similar tendency while DIDP-1 and DIDP-2 have better relative performance

Summary

- Optimally solving small instances (up to 75 or 100 locations):
  B&C and MILP are competitive and better than other approaches
- Finding good solutions for larger instances:
  DIDP is the best in the exact methods, and MILP is better than B&C
- The $O(n^2)$ dual bound for DIDP does not pay off

DIDP project page: https://didp.ai

Repository: https://github.com/domain-independent-dp/didp-rs

My thesis on DIDP won the **Outstanding Dissertation Award (Honourable Mention),** talk in the award session (15:30- tomorrow)

Branch-and-Cut (B&C) with ILP

Lazily add $\sum_{i\in S, j\in S\setminus\{i\}} x_{ij} \leq |S| - 1$ when the current solution contains a subtour consist of $S \subset N$ during branch-and-bound

$$\min \sum_{i\in N}\sum_{j\in N\setminus\{i\}}\sum_{k\in N\setminus\{i.j\}} c_{ijk}y_{ijk}$$

$$\sum_{j\in N\setminus\{i\}} x_{ij} = \sum_{j\in N\setminus\{i\}} x_{ji} = 1 \qquad \forall i \in N$$

$$x_{ij} = \sum_{k\in N\setminus\{i,j\}} y_{ijk} = \sum_{k\in N\setminus\{i,j\}} y_{kij} \qquad \forall i \in N, j \in N\setminus\{i\}$$

Eliminate subtours (cycles consist of a strict subset)

$$x_{ij} \in \{0,1\} \qquad \forall i \in N, j \in N\setminus\{i\}$$

$$y_{ijk} \in \{0,1\} \qquad \forall i \in N, j \in N\setminus\{i\}, k \in N\setminus\{i,j\}$$

Fischer et al. (2014), Oswin et al. (2017)