Improving Optimization Bounds using Machine Learning:

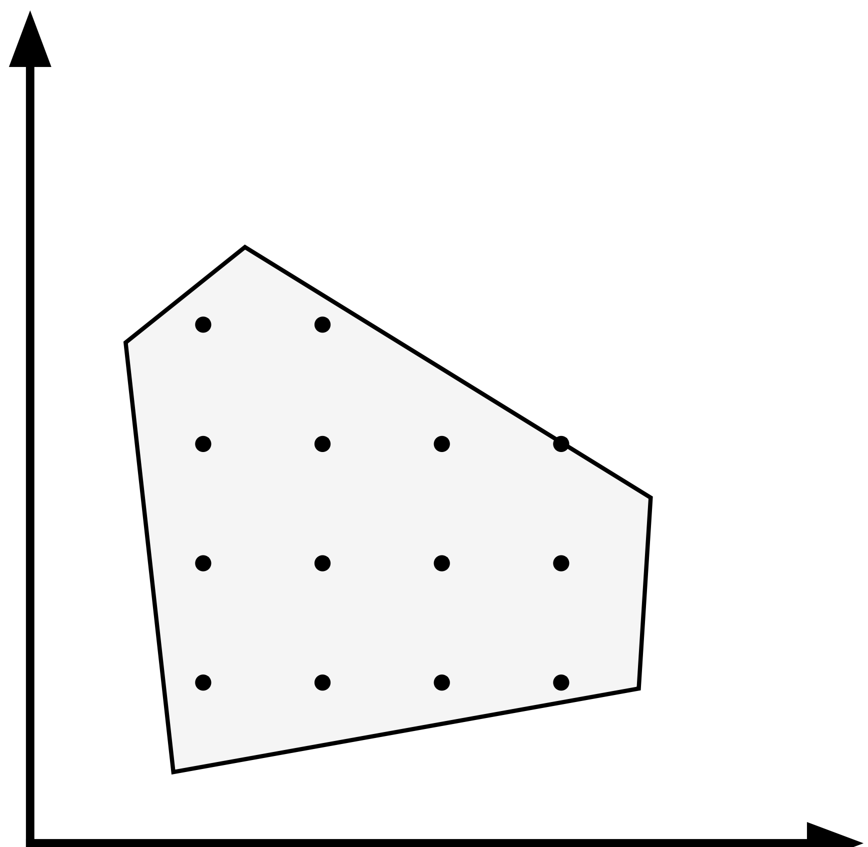
Decision Diagrams meet Deep Reinforcement Learning

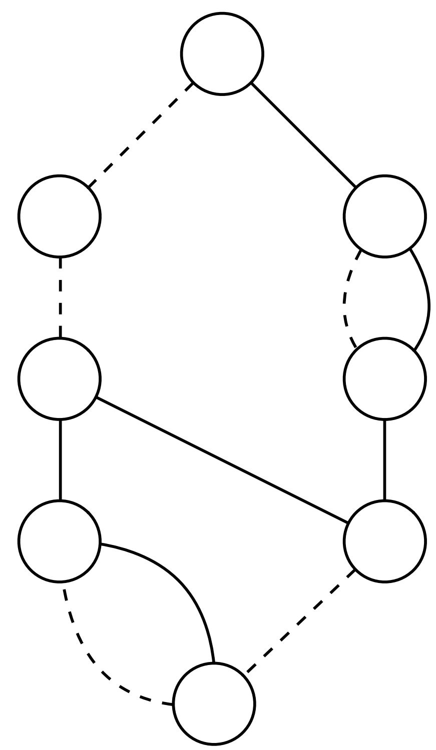
Quentin Cappart, *Emmanuel Goutierre,*
David Bergman, Louis-Martin Rousseau

Research question

Bounding mechanisms are critical in the design of scalable optimization solvers.

Inflexible bounds	Flexible bounds
Linear relaxation	Relaxed/Restricted decision diagrams

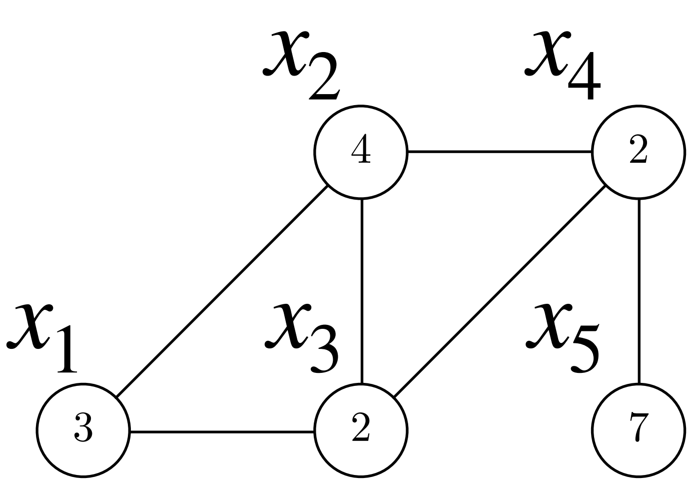




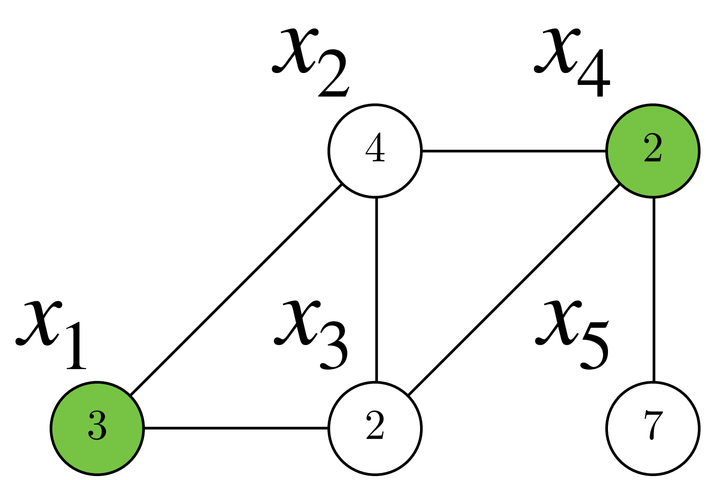
- Maximum width.
- Node merging.
- Variable ordering.

Running Example: Maximum Independent Set Problem

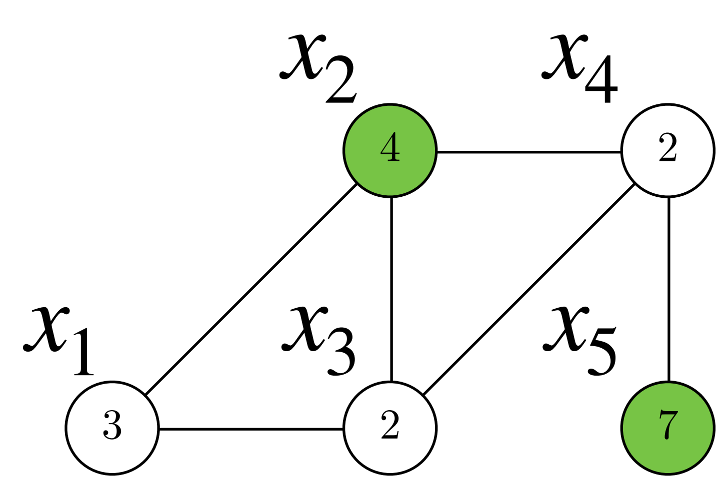
Given a graph, select the set of non adjacent vertices with the maximum weight.



Instance

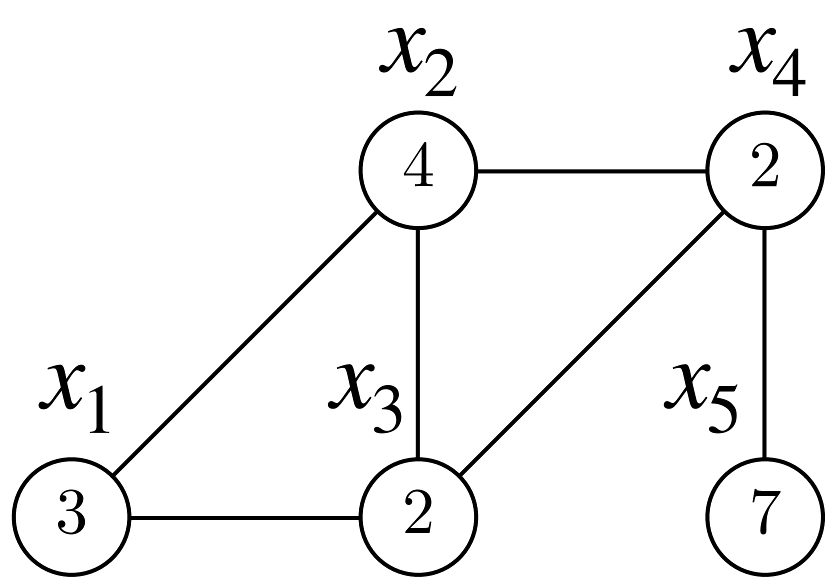


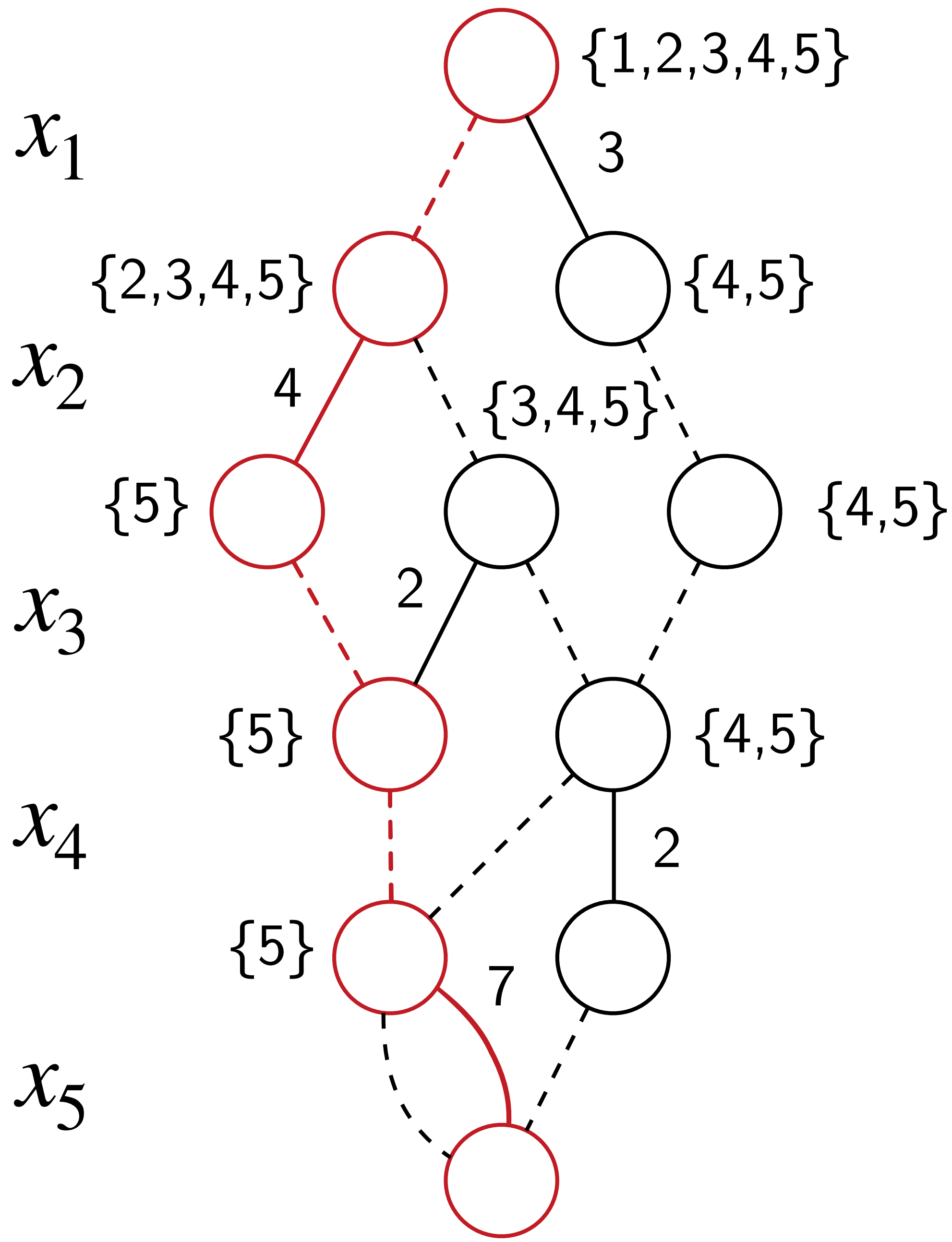
Weight = 5



Weight = 11
(Optimal)

Encoding MISP using decision diagrams





1. **Node state**: vertices that can be inserted.

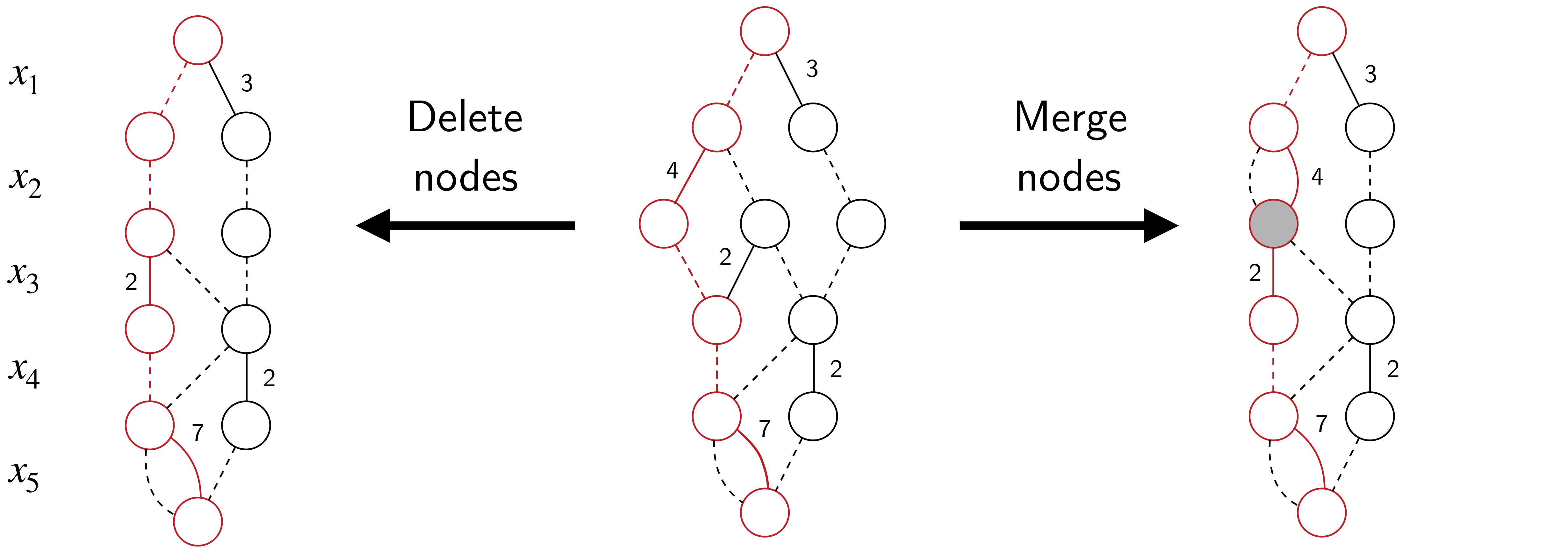
2. **Arc cost**: weight of the node, if inserted.

3. **Solution**: longest path in the diagram.

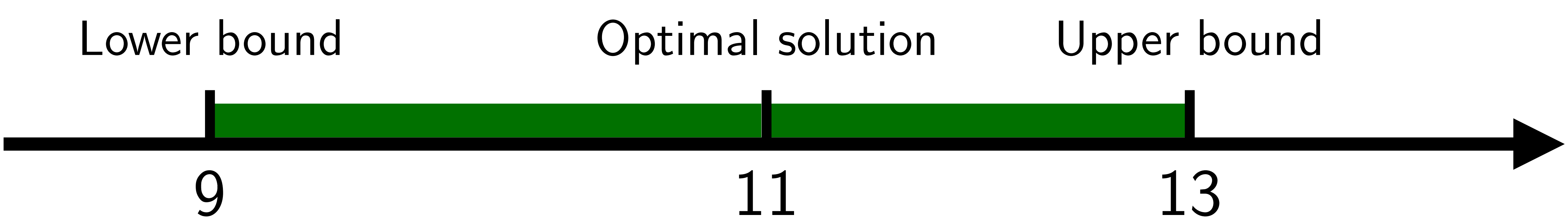
Solution = 4 + 7 = 11

Flexible bounds using decision diagrams (1/2)

Restricted DD   Exact DD   Relaxed DD

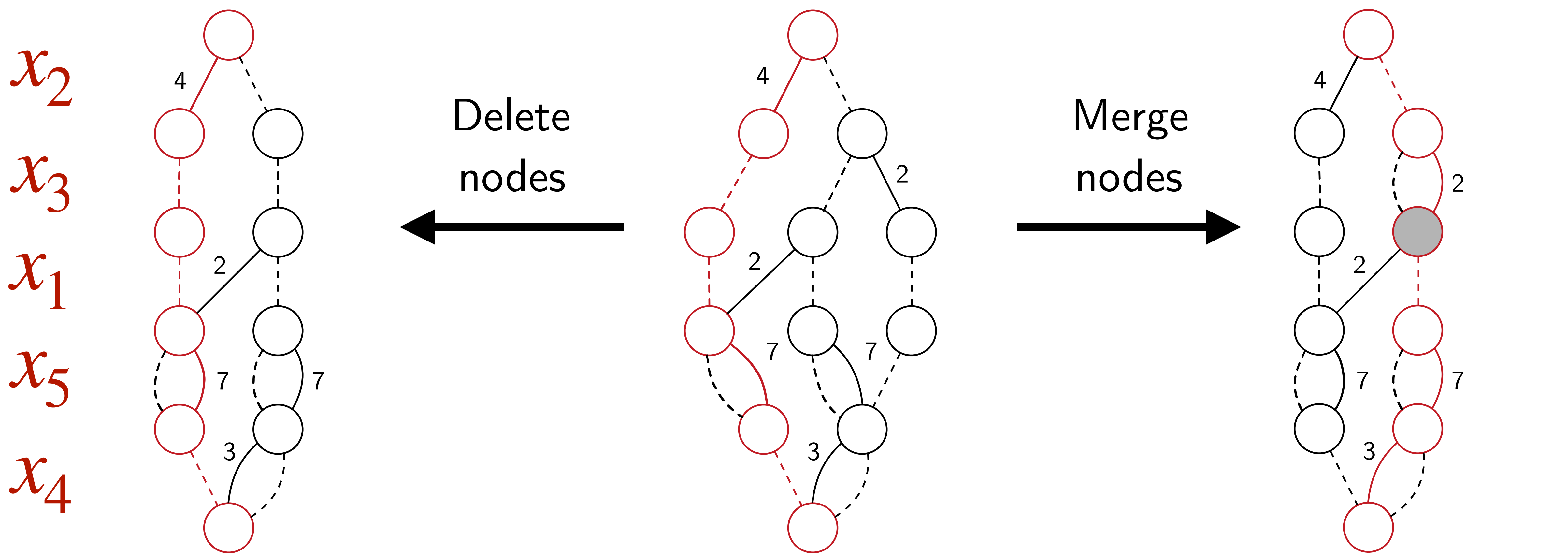


$2+7=9$ $4+7=11$ $4+2+7=13$



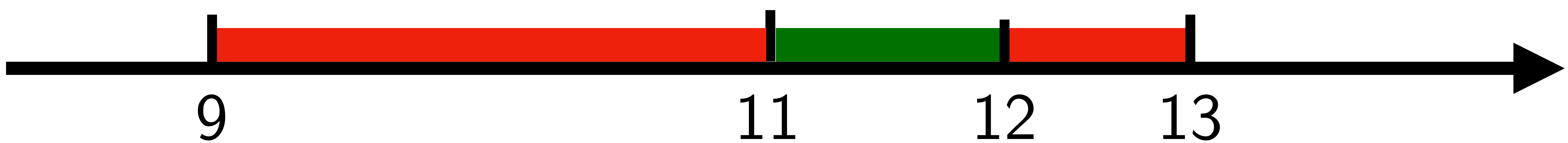
Flexible bounds using decision diagrams (2/2)

Restricted DD | Exact DD | Relaxed DD



$4 + 7 = 11$ | $4 + 7 = 11$ | $2 + 7 + 3 = 12$

Optimal solution



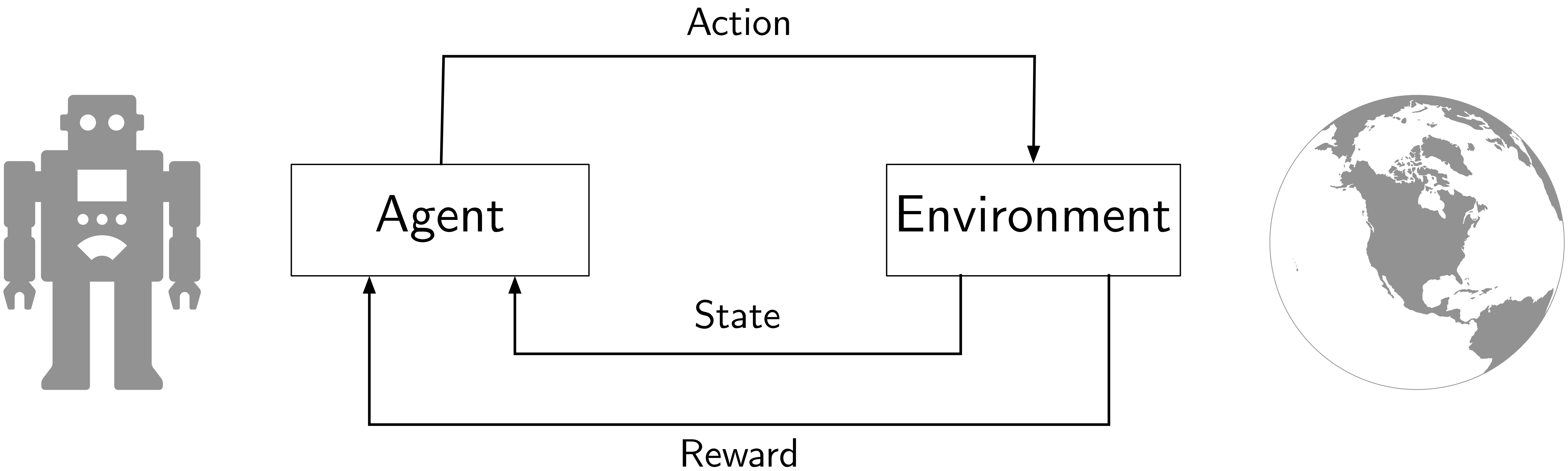
Improving a variable ordering is NP-hard

Variable ordering can have a huge impact on the bounds obtained.

But improving the variable ordering is NP-hard...

We propose a generic method based on Deep Reinforcement Learning.

Reinforcement learning in a nutshell (1/2)



1. The **agent** observes the **environment**.

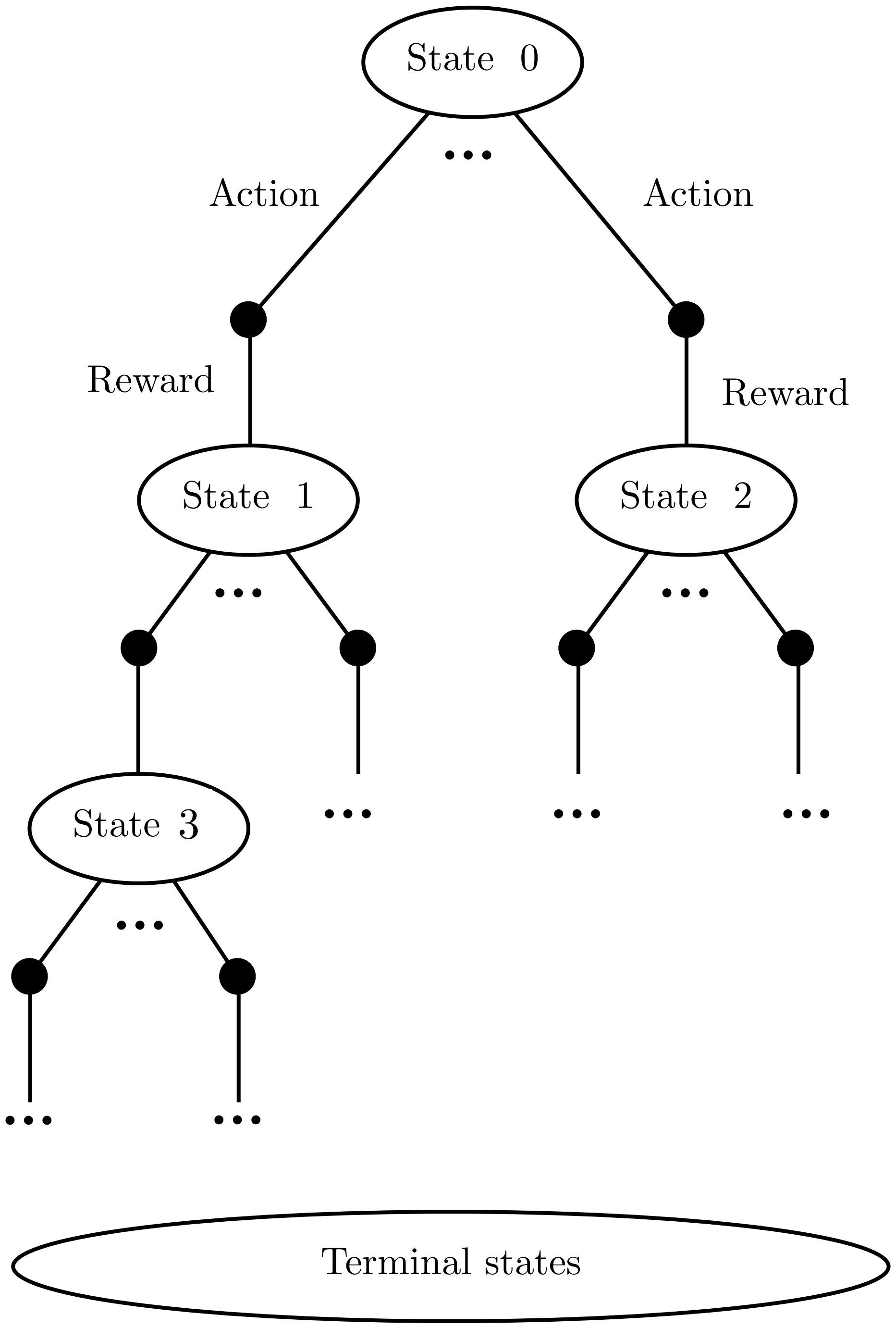
2. He chooses an **action**.

3. He gets a **reward** from it.

4. He moves to another **state**.

The goal is to maximize the sum of received rewards until a terminal state is reached.

Reinforcement learning in a nutshell (2/2)



Maximize the total reward.

How do we select the actions to do ?

In theory...

1. Compute an estimation of the quality of actions: **Q-values**.
2. Take the action having the best Q-value: **greedy policy**.
3. The **policy is optimal** if the Q-values are optimal.

In practice...

1. Search space to large to compute the optimal Q-values.
   Q-learning: iteratively update the Q-values through simulations.
2. Some states are never visited through the simulations.
   Deep Q-learning: approximate similar states using a deep network.

Reinforcement learning vs decision diagrams

Reinforcement Learning	**Decision Diagrams**
State Space	State Space
Action	Variable Selection
Reward function	Cost function
Transition function	Transition function
	Merging operation

There is a natural similarity !

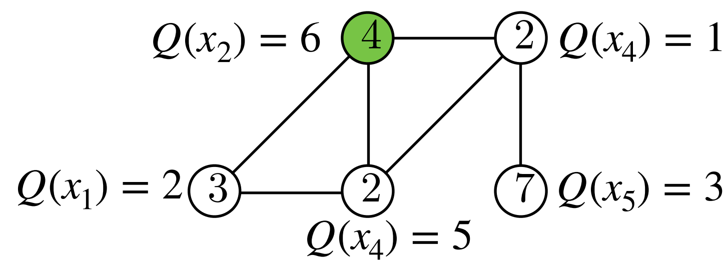
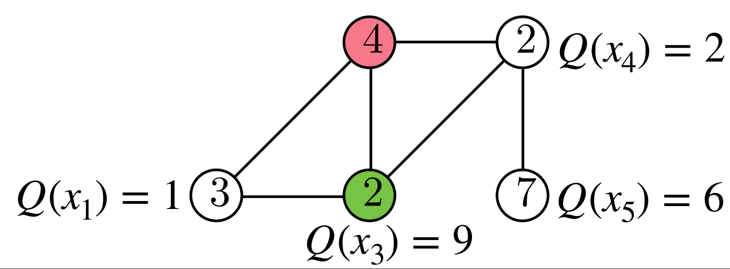
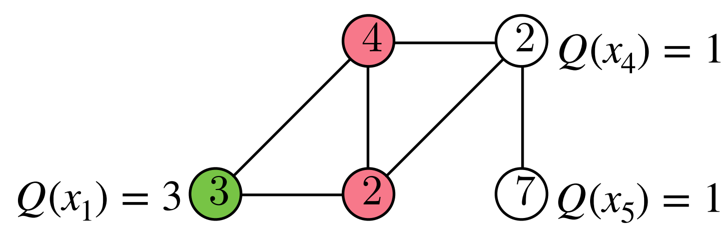
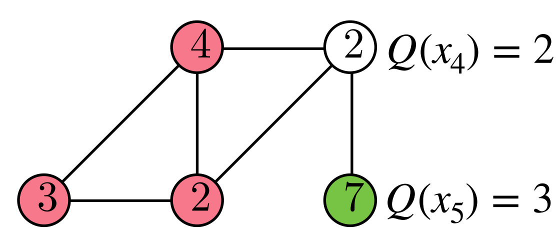
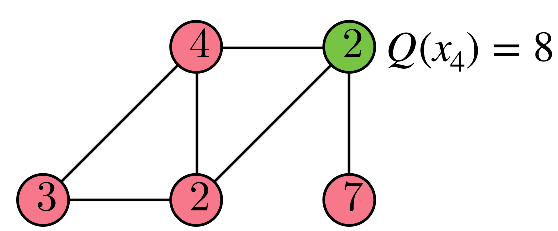
(Both are based on dynamic programming)

RL environment for decision diagrams

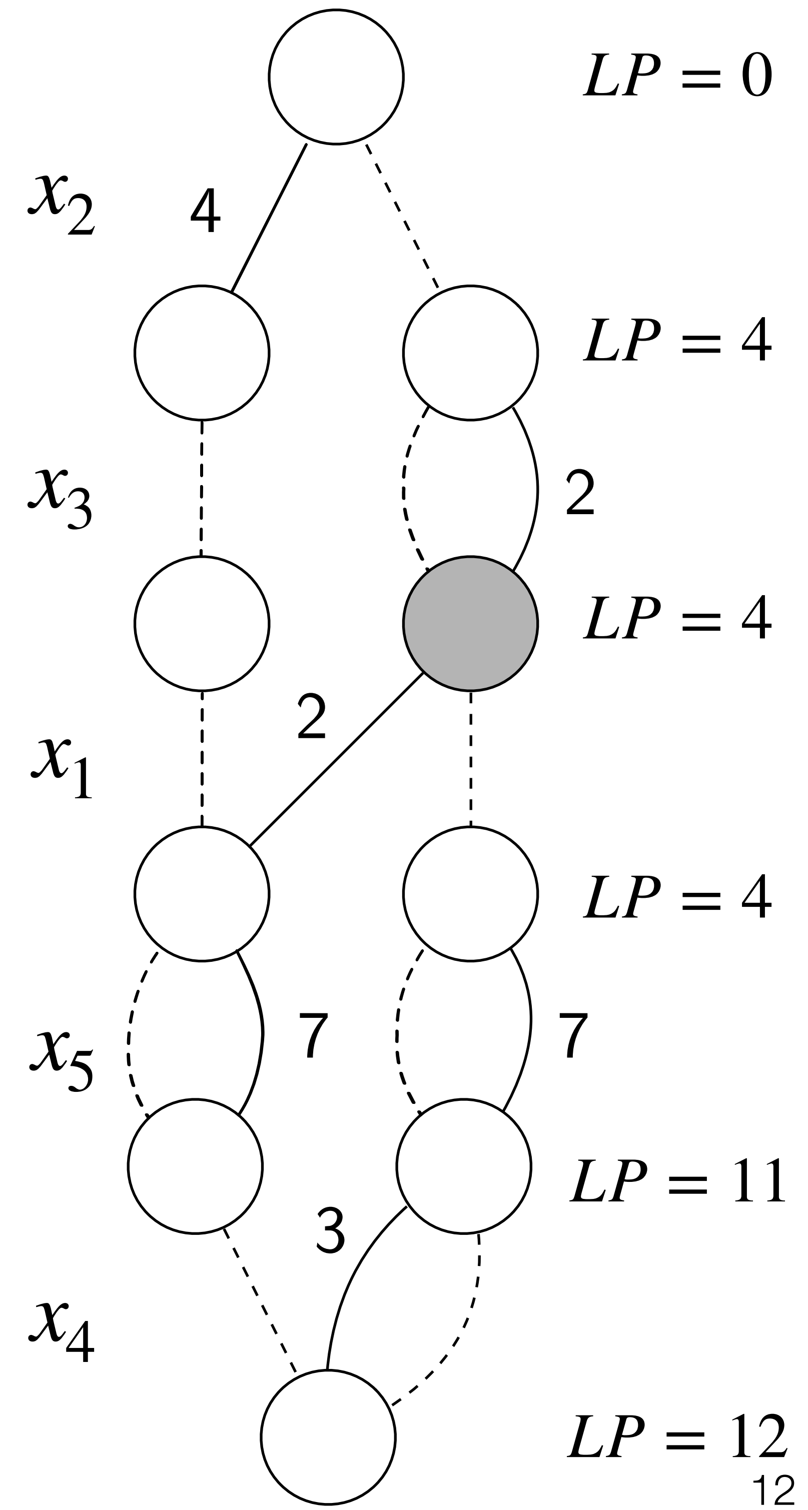
State	1. An ordered list of variables. 2. The DD currently built.
Action	Add a new variable in the DD.
Transition	Built the next layer of the DD using the selected variable.
Reward	Improvement in the new lower/upper bound (difference in the longest path).

For any COP that can be recursively encoded by a decision diagram.

Construction of the DD using RL

Sequence of states	Environment	Reward
• **State 1:** []		0
• **Action:** Inserting x_2		+ -4
• **State 2:** $[x_2]$		= -4
• **Action:** Inserting x_3		+ 0
• **State 3:** $[x_2, x_3]$		= -4
• **Action:** Inserting x_1		+ 0
• **State 4:** $[x_2, x_3, x_1]$		= -4
• **Action:** Inserting x_5		+ -7
• **State 5:** $[x_2, x_3, x_1, x_5]$		= -11
• **Action:** Inserting x_4		+ -1
• **State 6:** $[x_2, x_3, x_1, x_5, x_4]$ (Terminal state)		= -12

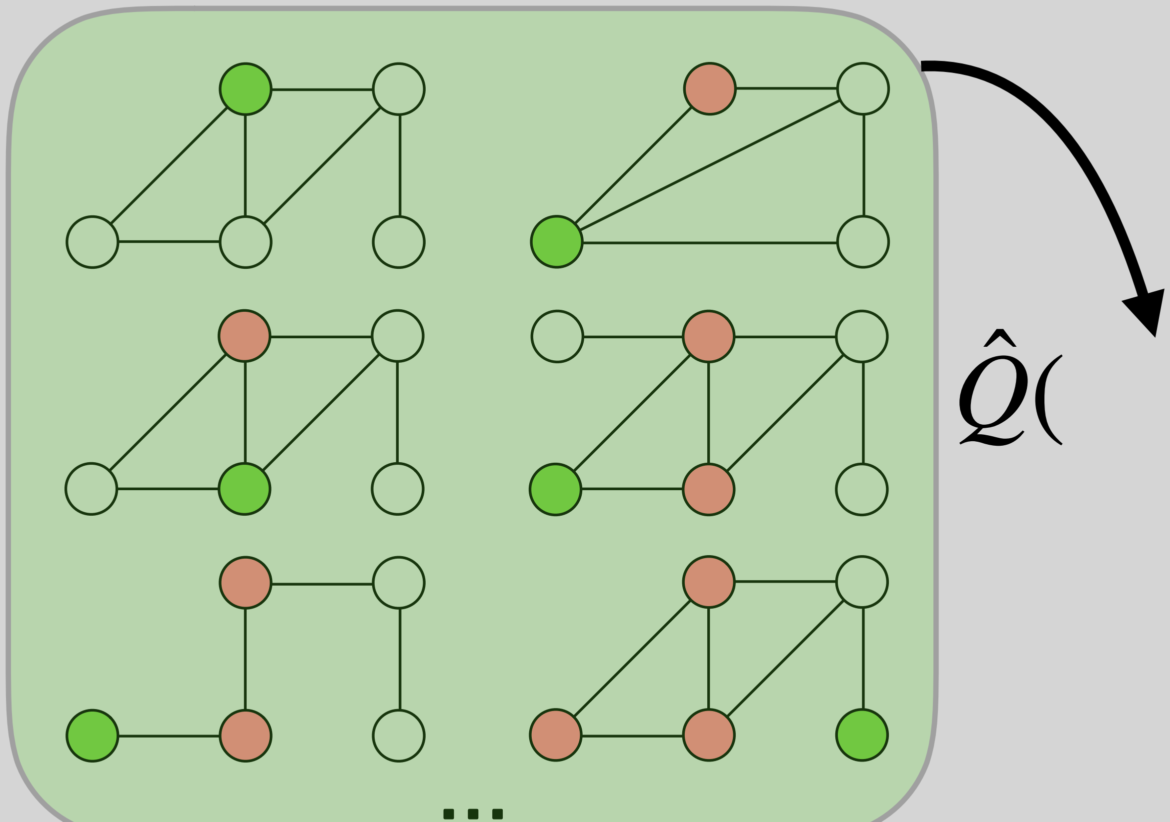
Current relaxed DD



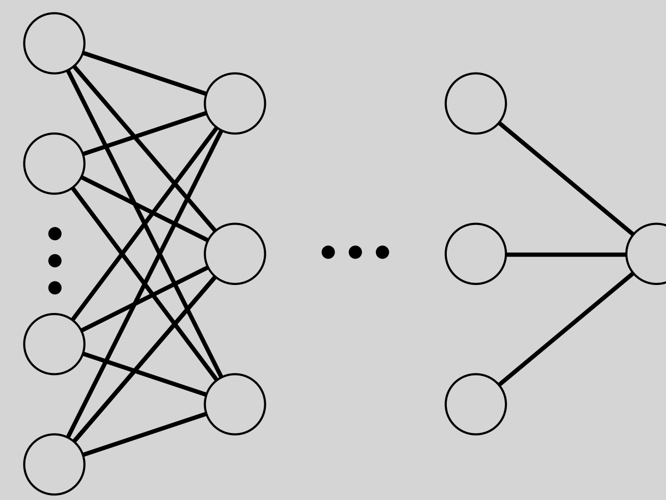
Computing the Q-values

$$Q(State, Action) \approx \hat{Q}(State, Action, \mathbf{Weight})$$

Training phase: parametrizing the weight

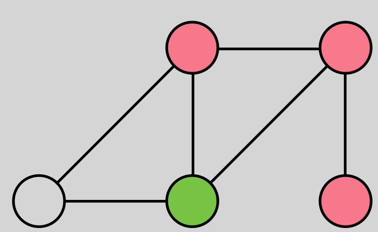


$\hat{Q}($ $,\mathbf{Weight}) =$



Evaluation: compute the estimated Q-value

$\hat{Q}($

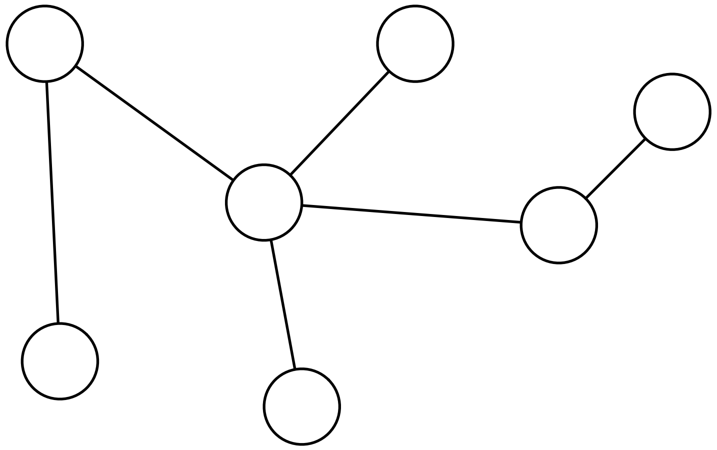


$,\mathbf{Weight}) = \mathbf{8}$

Training the model

1. Experiments on the **unweighted Maximum Independent Set Problem**.

$m = 1$

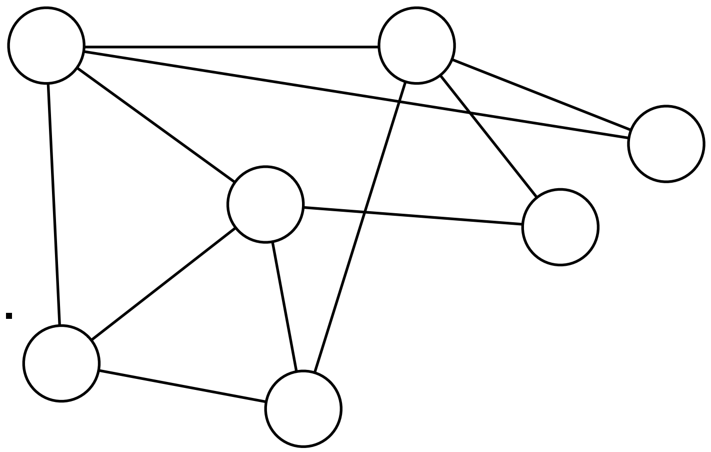


2. **Barabasi-Albert model**: real-world and scale-free graphs.

3. **Density known** by fixing the attachment parameter.

4. Graphs between **90 and 100 nodes**.

$m = 2$



5. **Maximal width for training is 2**.

6. **5000 randomly generated** BA graphs and periodically **refreshed**.

7. **Independent models** for relaxed and restricted DDs.

Main assumption:

the nature of the graphs we want to access is known.

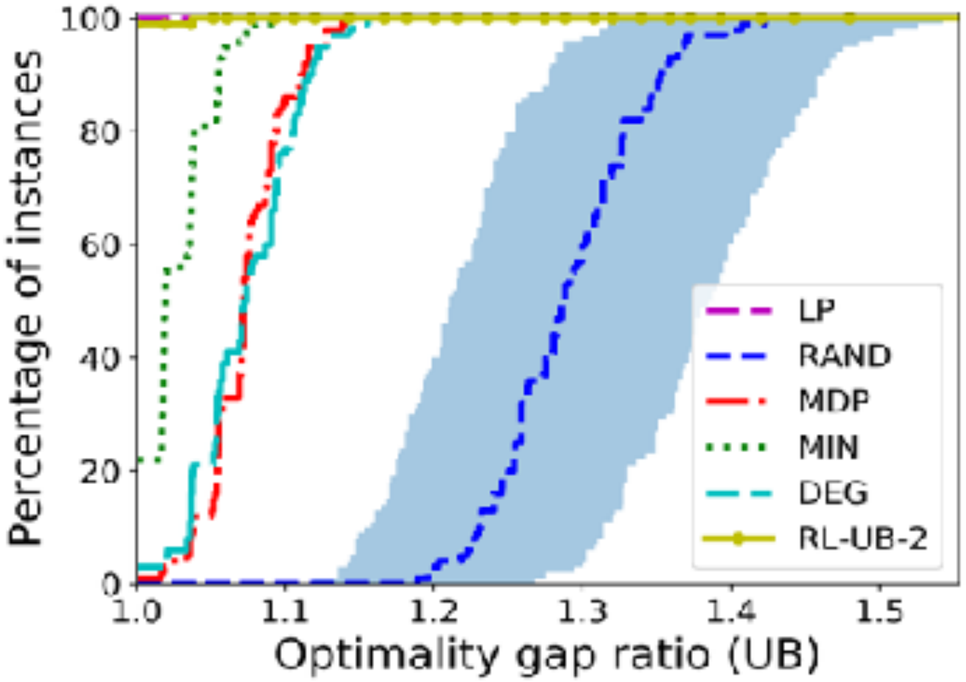
Experimental setup

1. Comparison with common heuristics (random, MPD, **min-in-state** and **vertex-degree**).
2. Comparison with **linear relaxation** (only with relaxed DDs).
3. **Width of 100** for relaxed DDs and **width of 2** for restricted DDs.
4. Graphs between **90 and 100 nodes**.
5. Different configurations for the attachment parameter (**2**, **4**, **8** and **16**).
6. Tested on **100 new random graphs**.
7. Compared with the **optimality gap** using **performance profiles**.

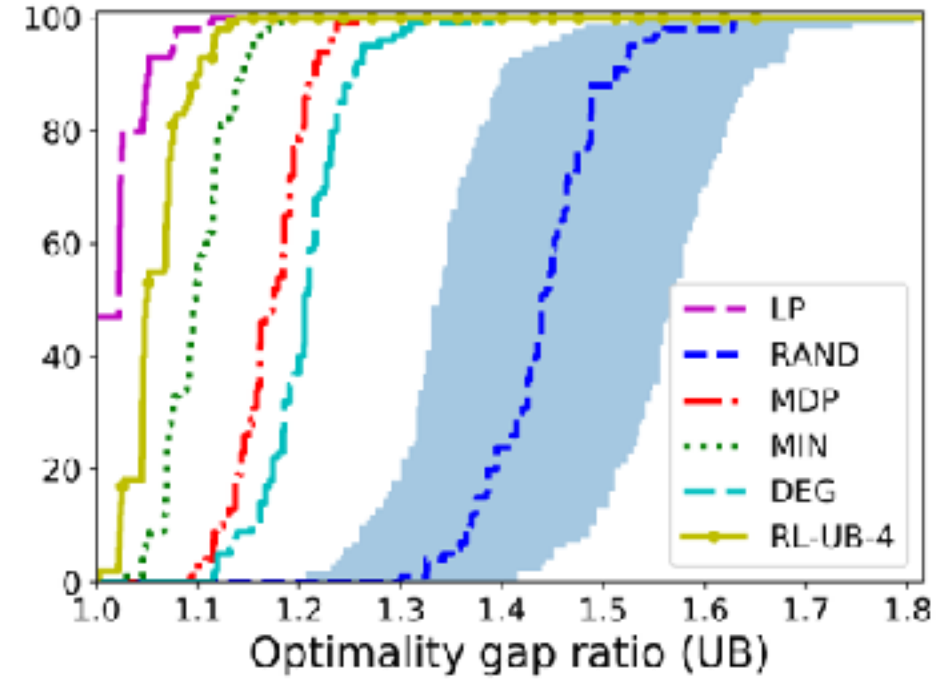
Other configurations are then tested.

Experiments for relaxed DDs (width = 100)

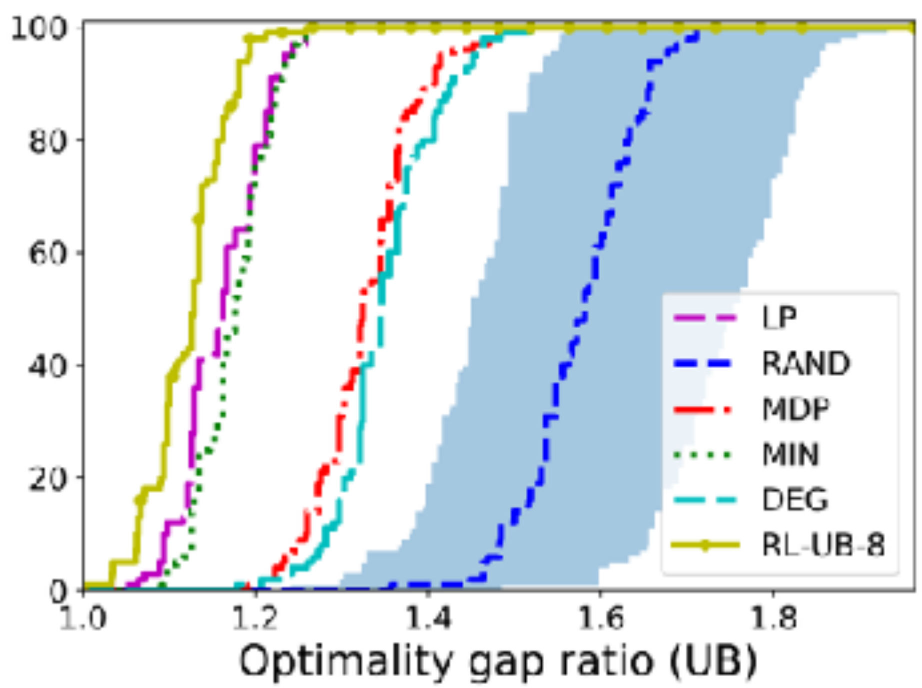
$m = 2$



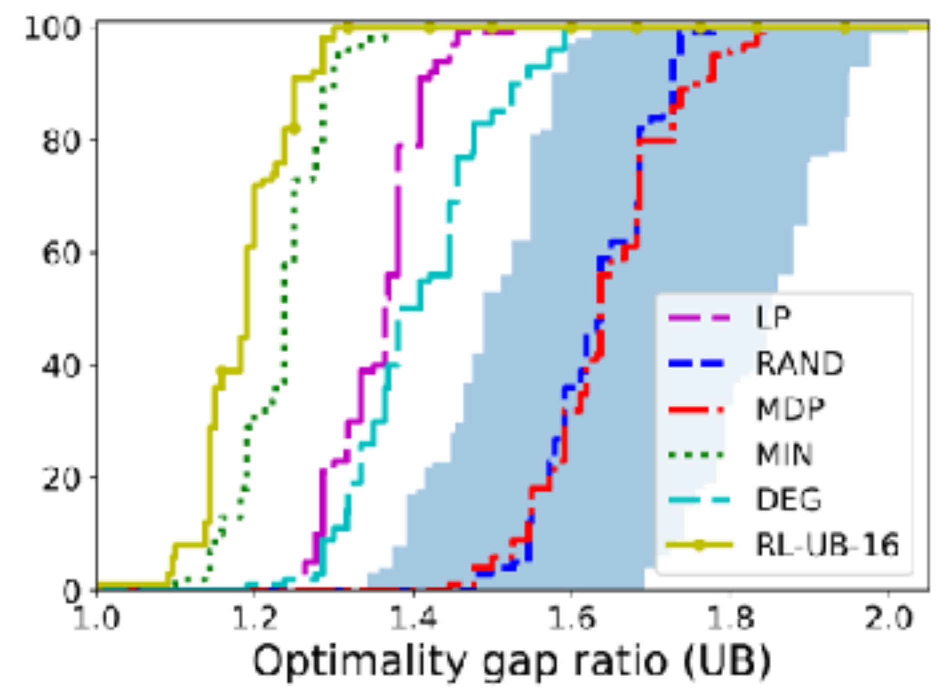
$m = 4$



$m = 8$

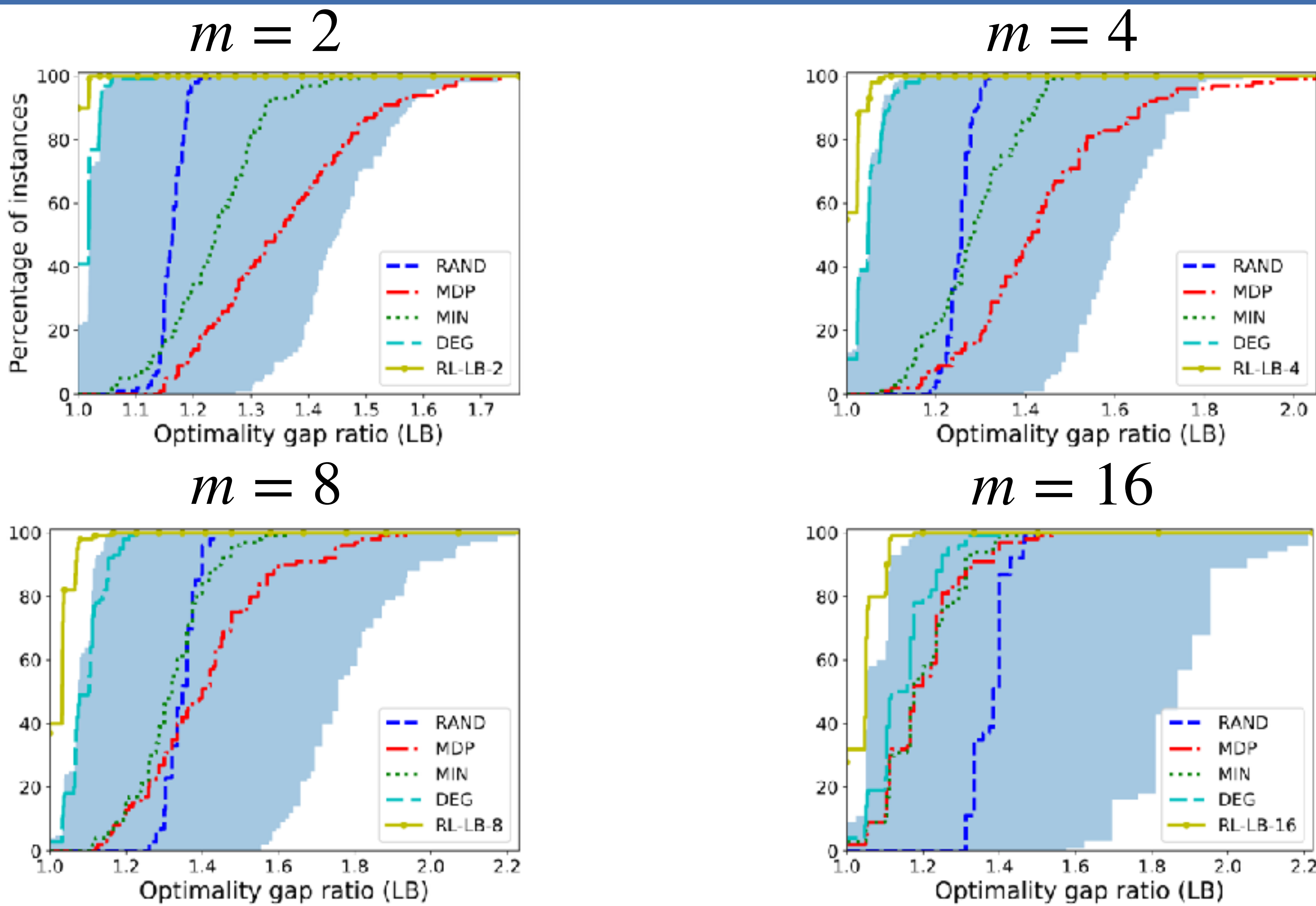


$m = 16$



RL is the best ordering and is better than LP for denser graphs.

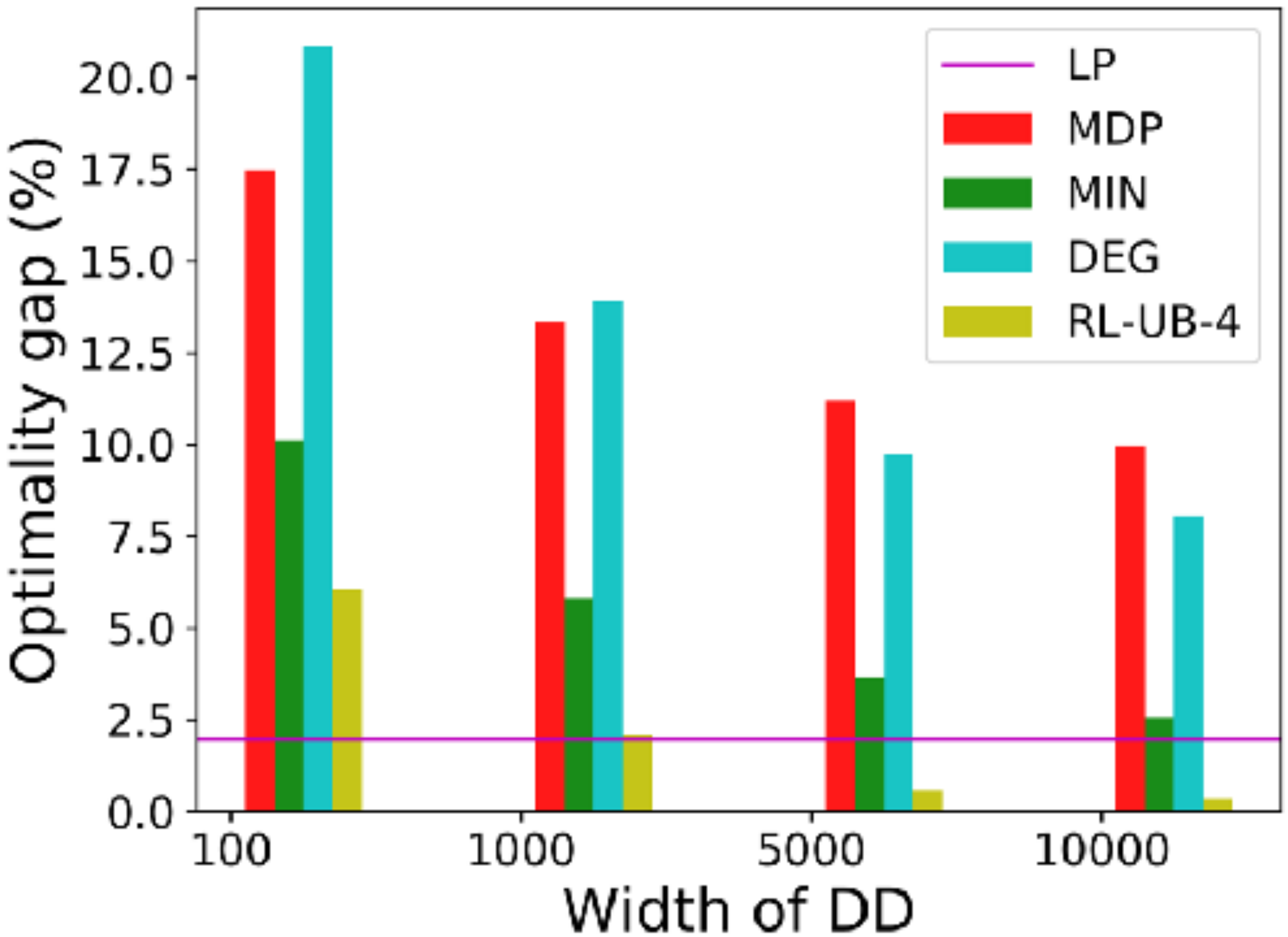
Experiments for restricted DDs (width = 2)

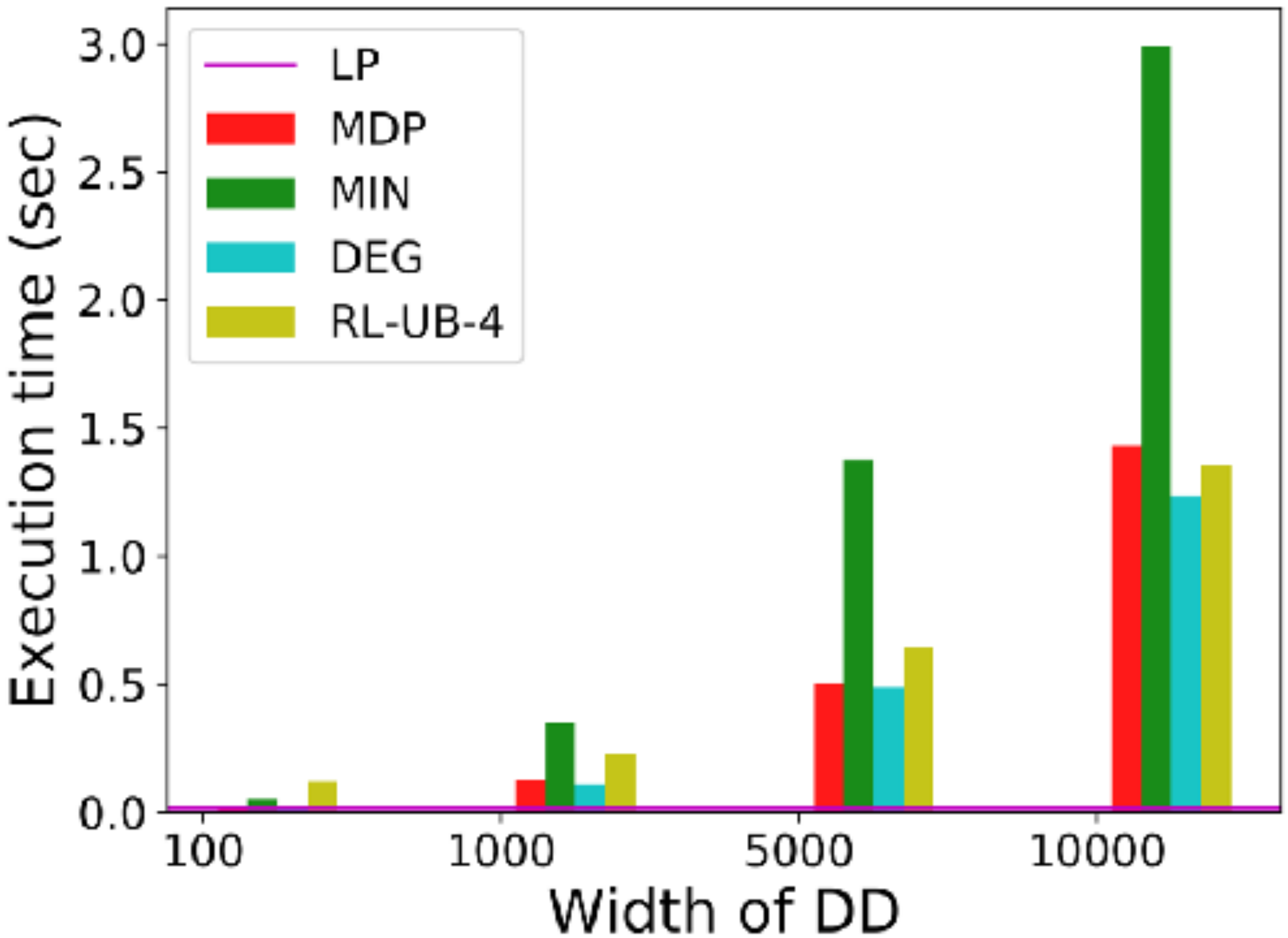


RL gives the best ordering in almost all situations.

Increasing the width for relaxed DDs

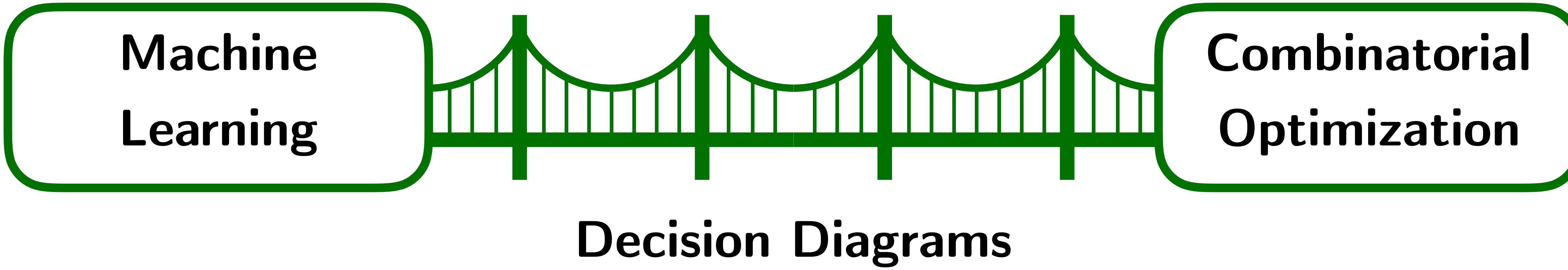
Training still done with a width of 2.





The model is robust when the width increases and the execution time remains acceptable.

Conclusion and perspectives



Contributions and results:

1. A **generic approach** based on DDs for learning flexible bounds.
2. **Better performances** than classical approaches on the MISP.
3. **Robust approach** for larger graphs and width.

Perspectives and future work:

1. **Data augmentation** for real-life instances.
2. Application to **other problems**.
3. Improvement using **other algorithms or approximators.**
4. Application to **other fields** (constraint programming, planning, etc.)

Improving Optimization Bounds using Machine Learning

quentin.cappart@polymtl.ca

arxiv.org/abs/1809.03359 <To replace with the AAAI link>

github.com/qcappart/learning-DD

POLYTECHNIQUE MONTRÉAL

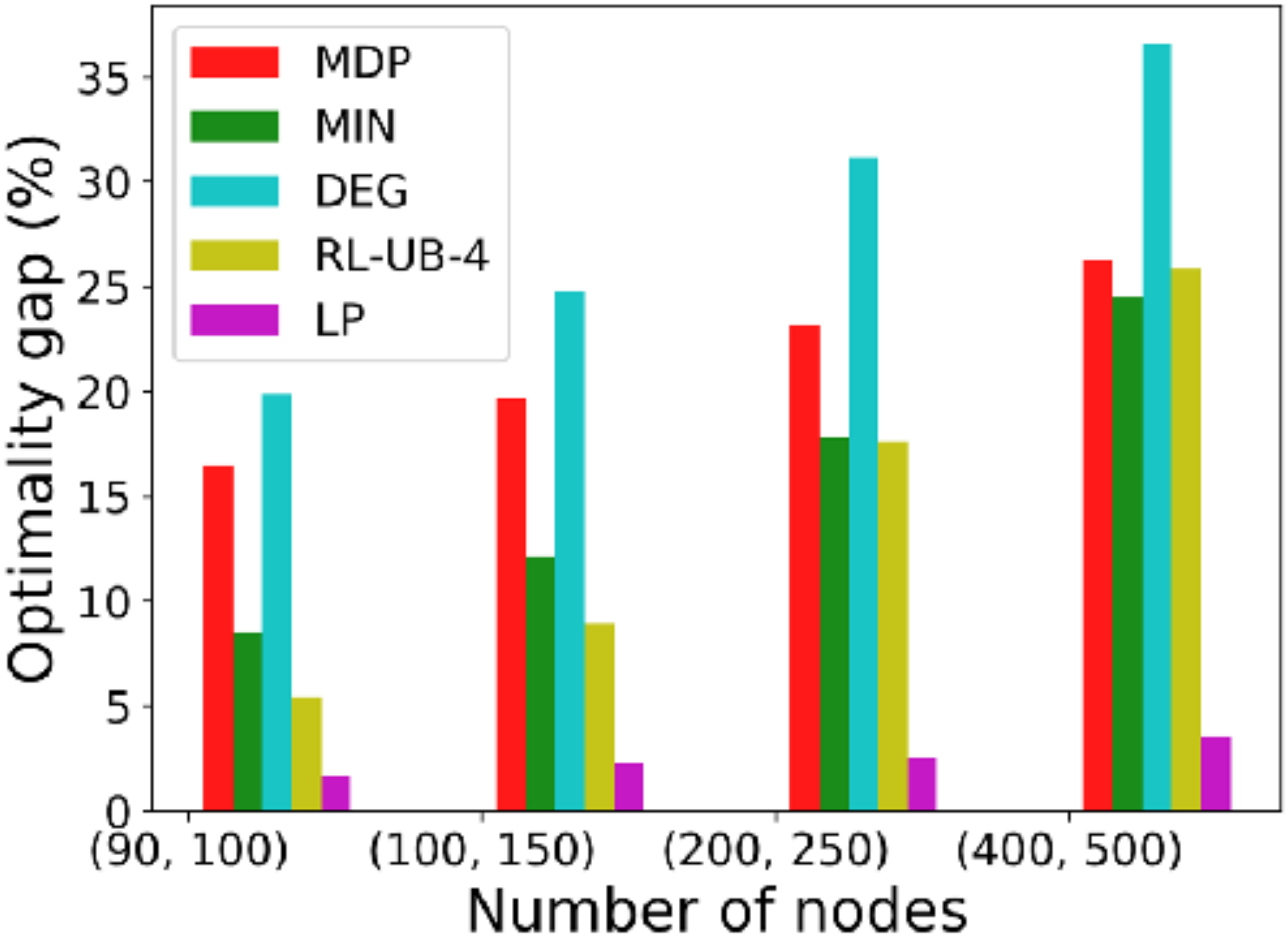
UCONN SCHOOL OF BUSINESS

ÉCOLE POLYTECHNIQUE UNIVERSITÉ PARIS-SACLAY

Increasing the graph size (width = 100)

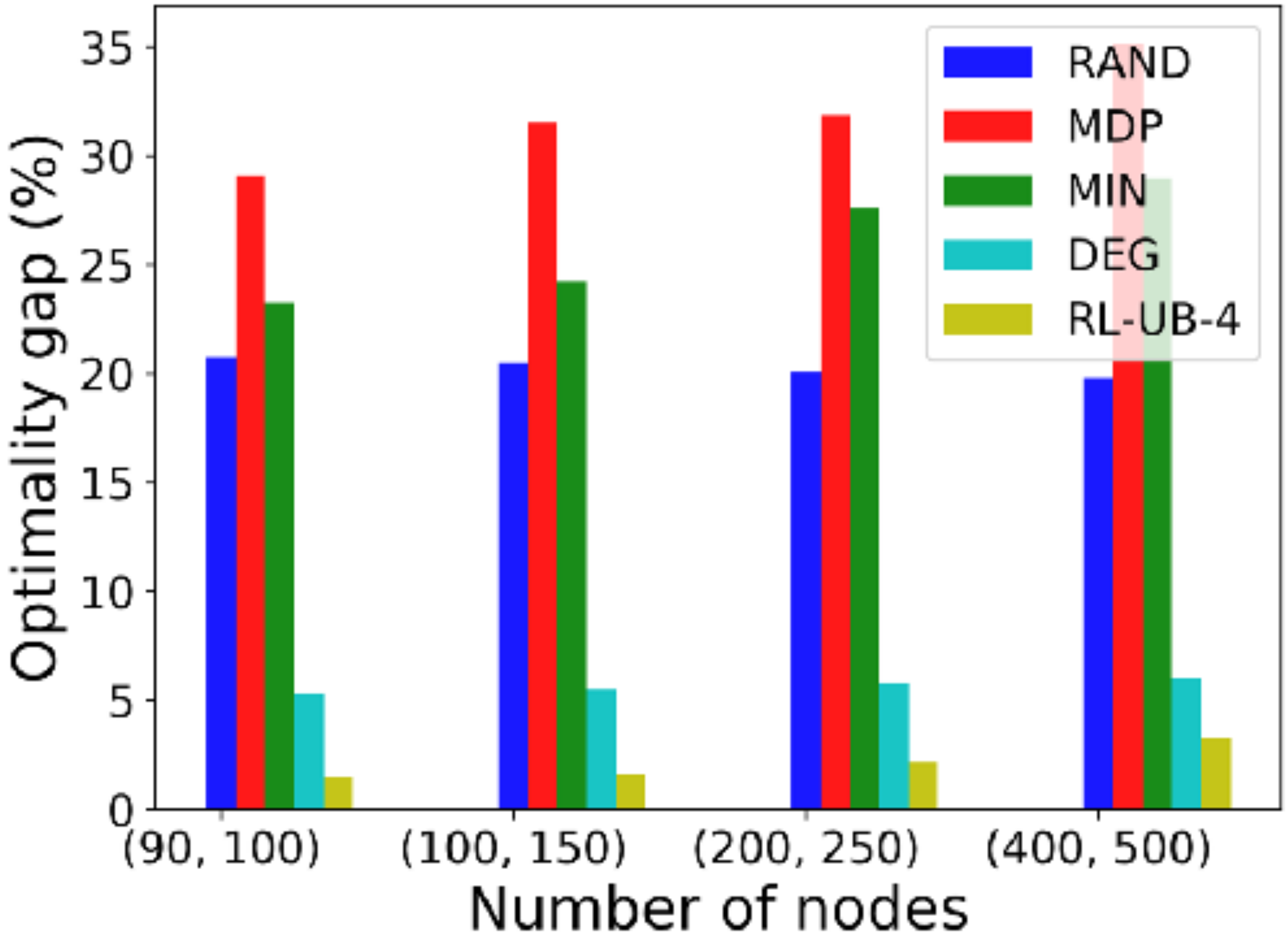
Training still done with graphs of 90 to 100 nodes.

Relaxed DDs



Fairly robust.

Restricted DDs

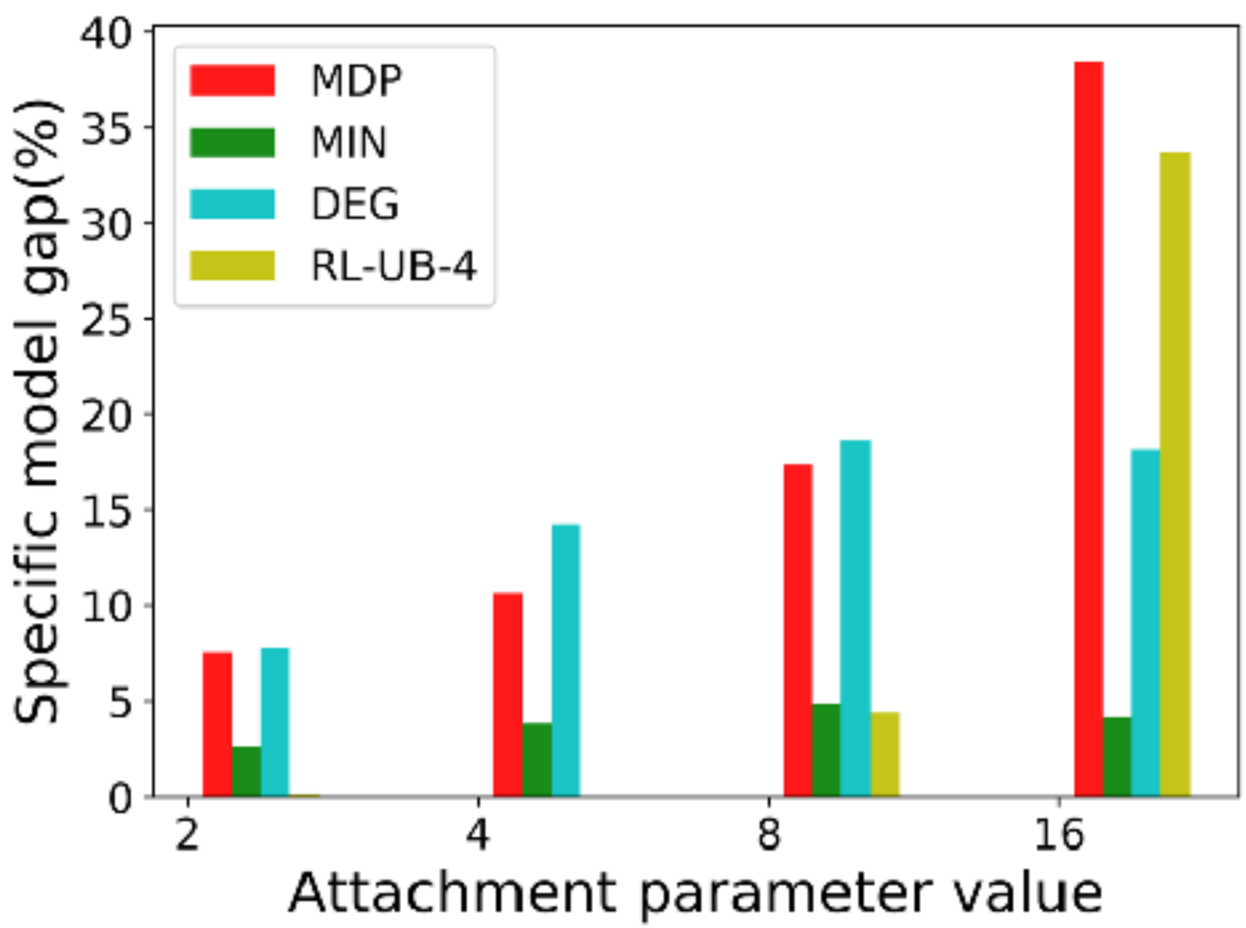


Strongly robust.

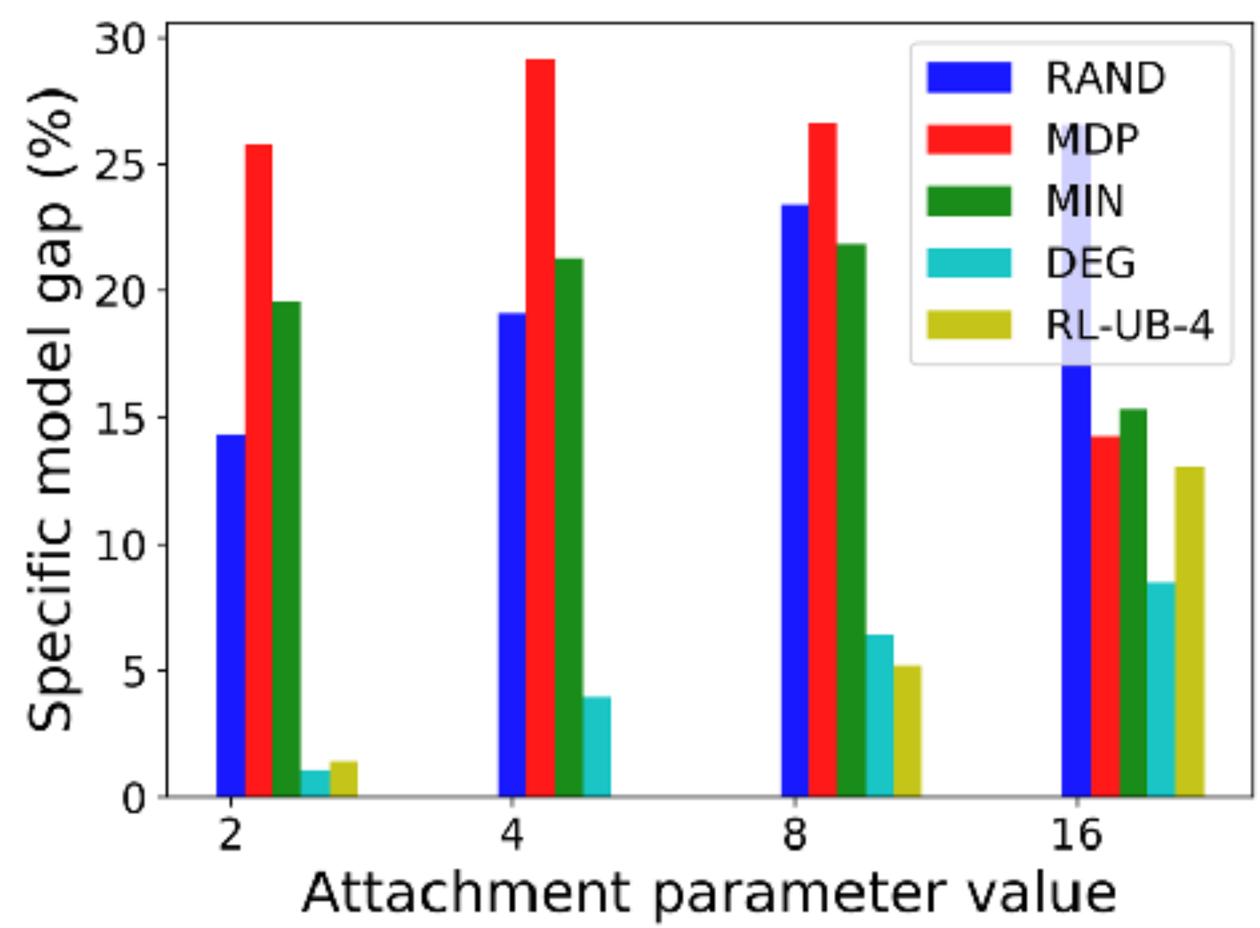
Modifying the distribution (width = 100)

Training done with an attachment parameter of 4.

Relaxed DDs



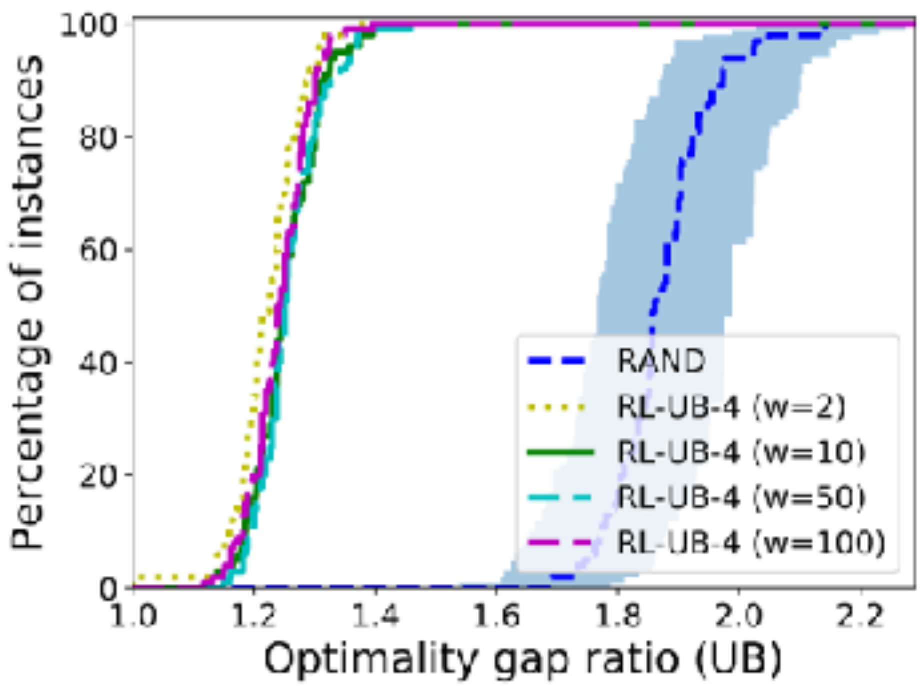
Restricted DDs



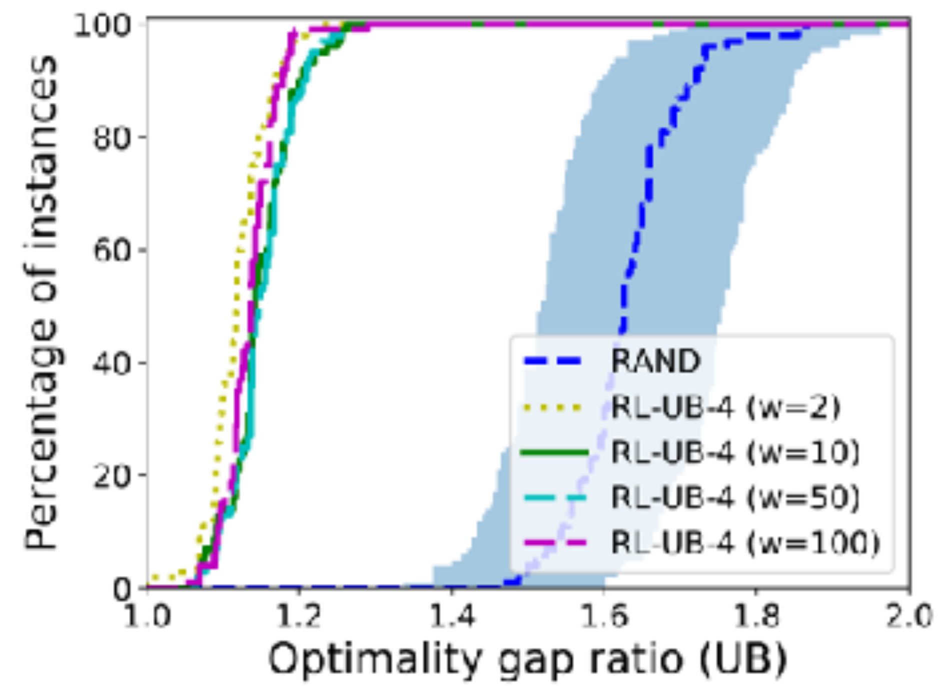
Important to know the distribution of the graphs we want to access.

Impact of the width used during training

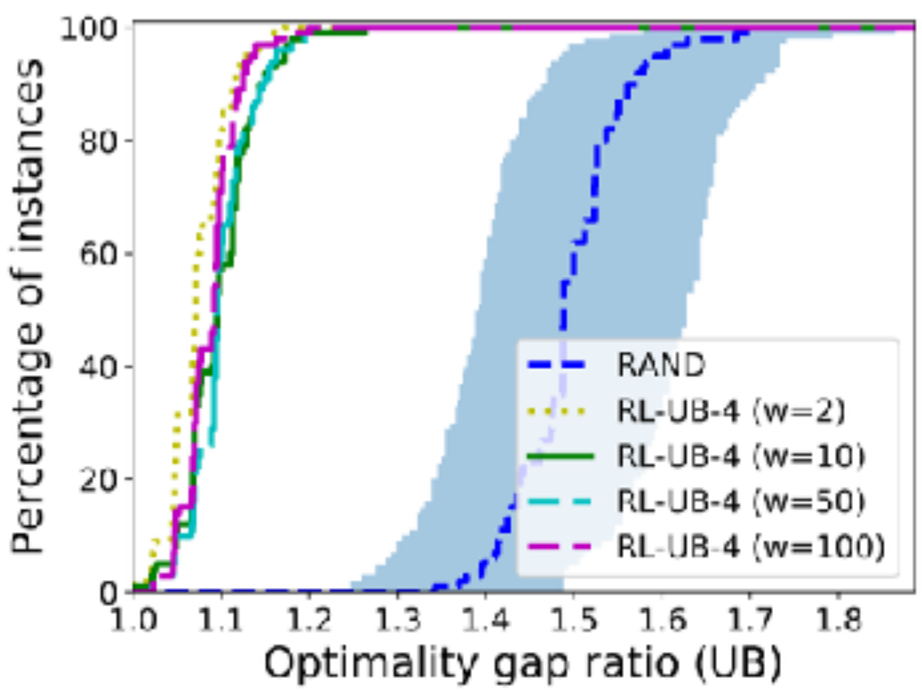
Testing width = 2



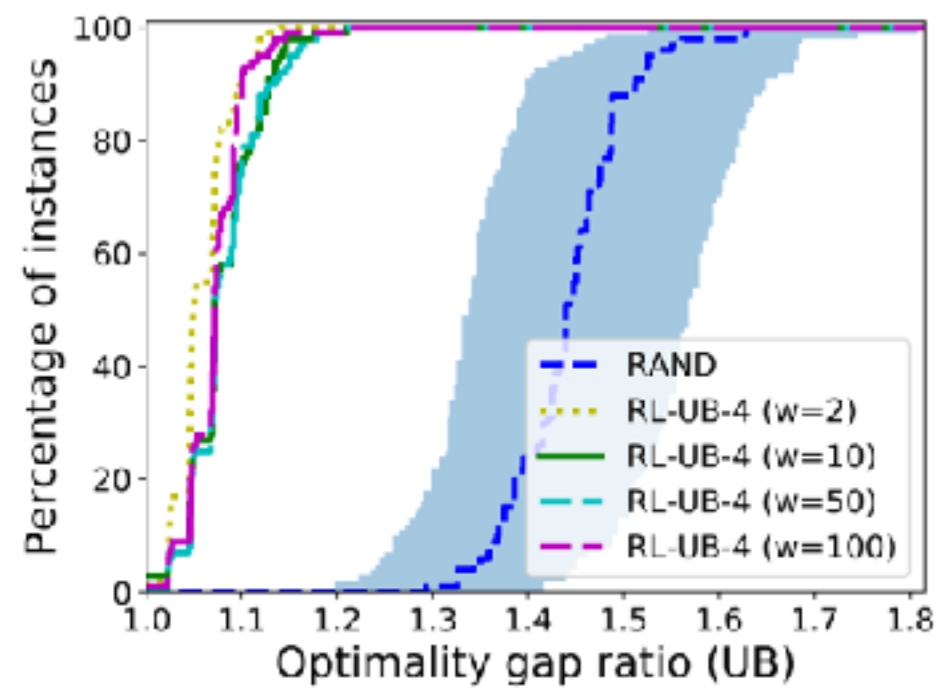
Testing width = 10



Testing width = 50



Testing width = 100

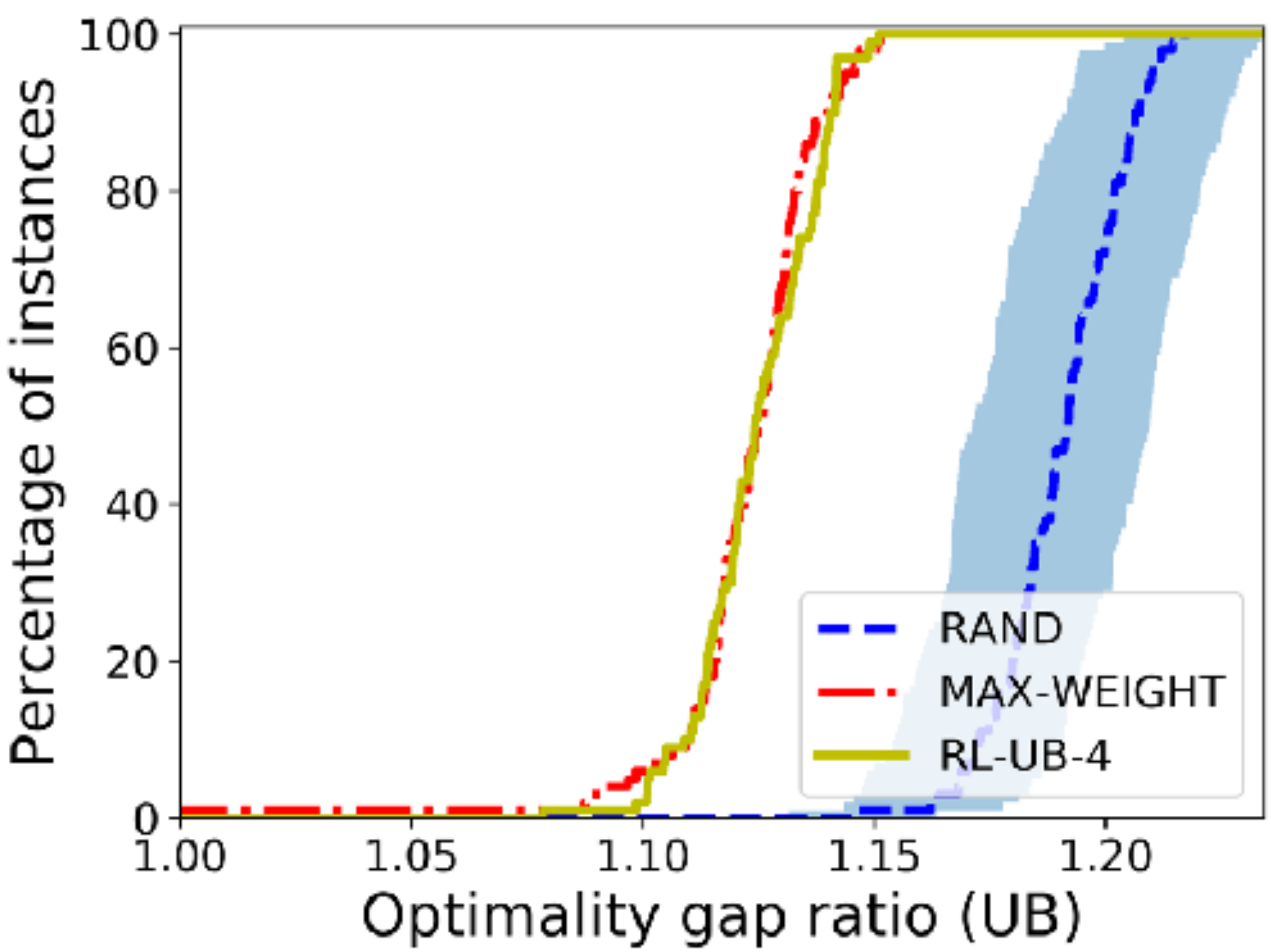


Ordering independent of the width chosen during the training.

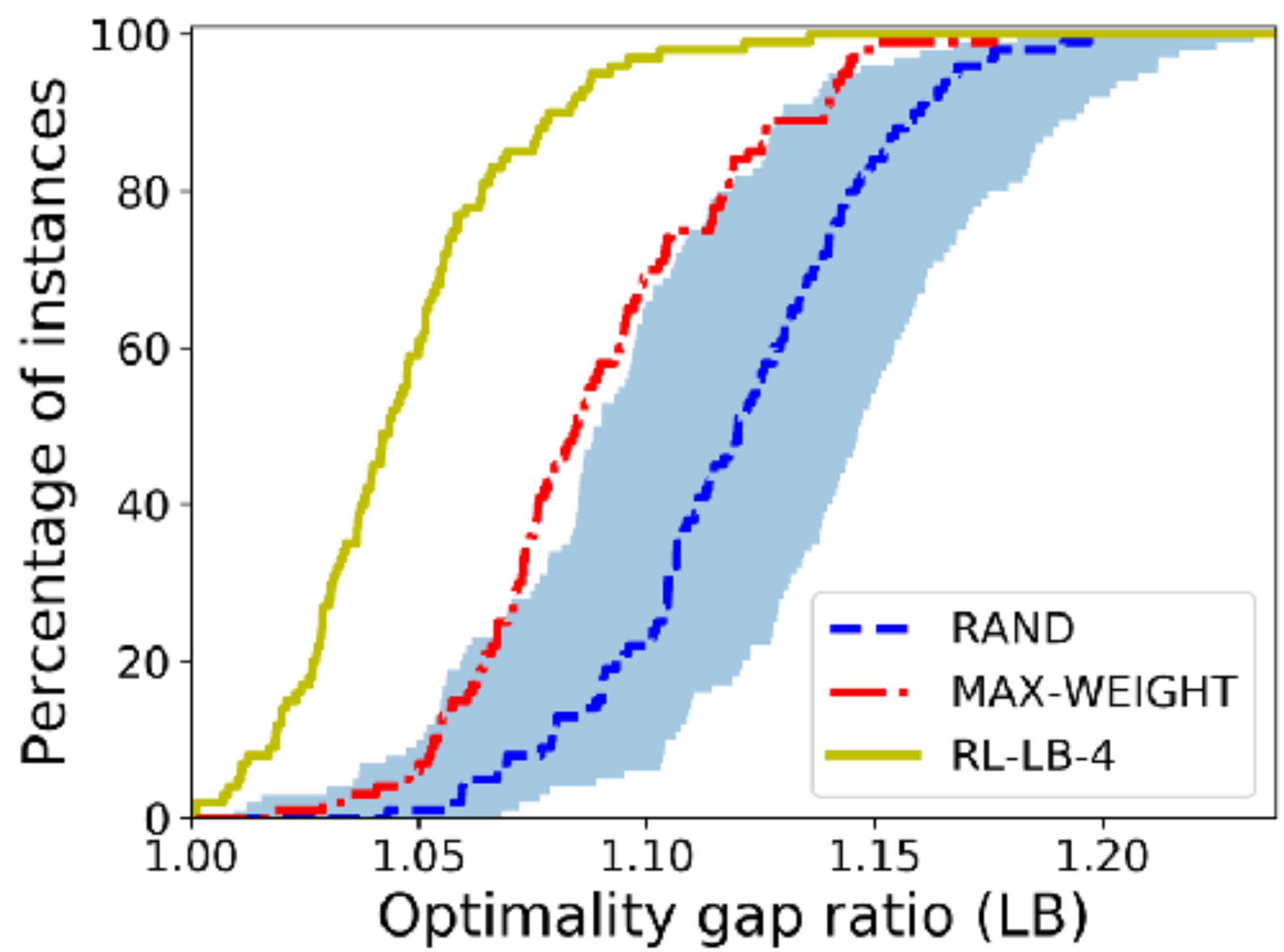
Application to Maxcut problem (work in progress)

Given a graph, select a set of nodes such that the weighted cut with the set of non selected nodes is maximized.

Relaxed DDs (width = 100)



Restricted DDs (width = 2)



Promising results but more difficult than the MISP.