Problem Solving and Search in Artificial Intelligence

Algorithm Selection

Nysret Musliu
Database and Artificial Intelligence Group
Institut of Logic and Computation, TU Wien

Motivation

- Usually several search algorithms are available for solving a particular problem
- No free lunch theorem

"…for any algorithm, any elevated performance over one class of problems is offset by performance over another class" [1]

"any two algorithms are equivalent when their performance is averaged across all possible problems" [2]

How to select the best algorithm for a specific instance?

[1] David Wolpert, William G. Macready: No free lunch theorems for optimization. IEEE Transac. Evolutionary Computation 1(1): 67-82 (1997)
[2] Wolpert, D.H., and Macready, W.G. (2005) "Coevolutionary free lunches," IEEE Transac. on Evolutionary Computation, 9(6): 721-735

Algorithm selection (Rice's framework)

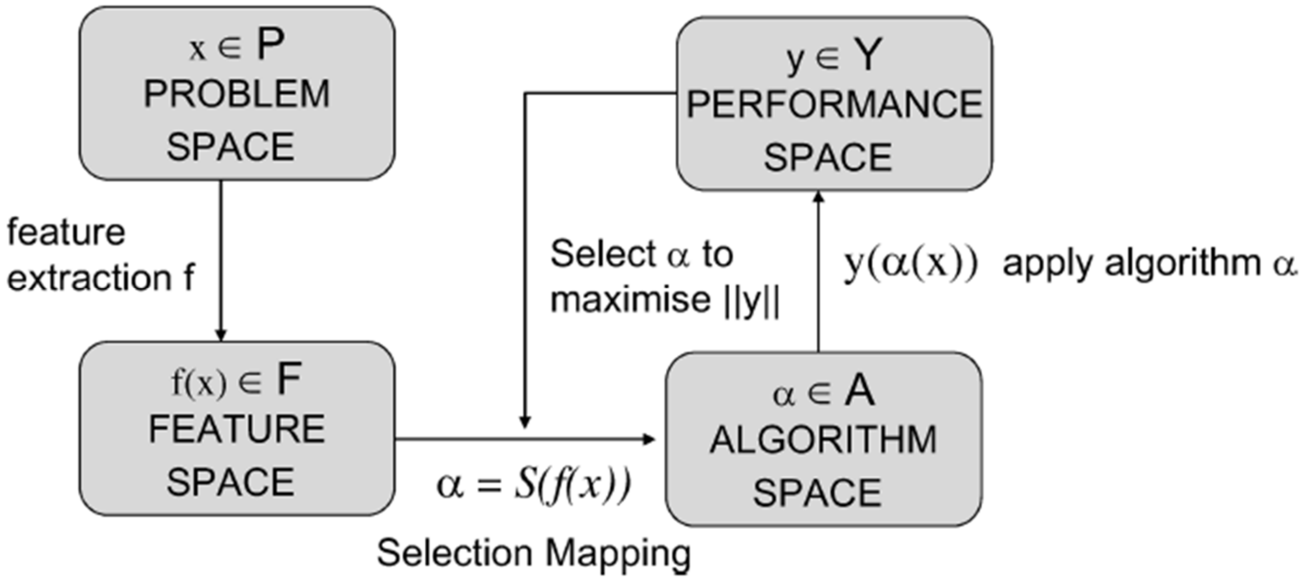


Figure taken from [9]

[8] John R. Rice: The Algorithm Selection Problem. Advances in Computers 15: 65-118 (1976)

[9] Kate Smith-Miles: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1): (2008)

Algorithm selection

Input (see [8] and [9]):

- Problem space P that represents the set of instances of a problem class
- A feature space F that contains measurable characteristics of the instances generated by a computational feature extraction process applied to P
- Set A of all considered algorithms for tackling the problem
- The performance space Y represents the mapping of each algorithm to a set of performance metrics

Problem:

For a given problem instance *x E P, with features f(x) E F, find the selection mapping S(f(x)) into algorithm space , such that the selected algorithm a E A maximizes the performance mapping y(a(x)) E Y*

[8] John R. Rice: The Algorithm Selection Problem. Advances in Computers 15: 65-118 (1976)
[9] Kate Smith-Miles: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1): (2008)

Algorithm selection

- An important issue is the selection of appropriate features
  - Example: Selection of sorting algorithms based on features ([10]):
    - Degree of pre-sortedness of the starting sequence
    - Length of sequence
- A supervised machine learning approach can be used to select the algorithm to be used based on features of the input instance
- A training set with instances (and their features) and best performing algorithm should be provided to the supervised machine learning algorithms to train them

[9] Kate Smith-Miles: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1): (2008)
[10] Guo, H. 2003. Algorithm selection for sorting and probabilistic inference: A machine learning-based approach. Ph.D. dissertation, Kansas State University.

Algorithm selection for sorting [10]

- P=43195 instances of random sequences of different sizes and complexities
- A=5 sorting algorithms (InsertionSort, ShellSort, heapSort, mergeSort, QuickSort)
- Y=algorithm rank based on CPU time to achieve sorted sequence
- F=3 measures of presortedness and length of sequences (size)
- Machine learning methods: C4.5, Naïve Bayes, Bayesian network learner

Different other examples are given in [9]

[10] Guo, H. 2003. Algorithm selection for sorting and probabilistic inference: A machine learning-based approach. Ph.D. dissertation, Kansas State University.

Other approaches

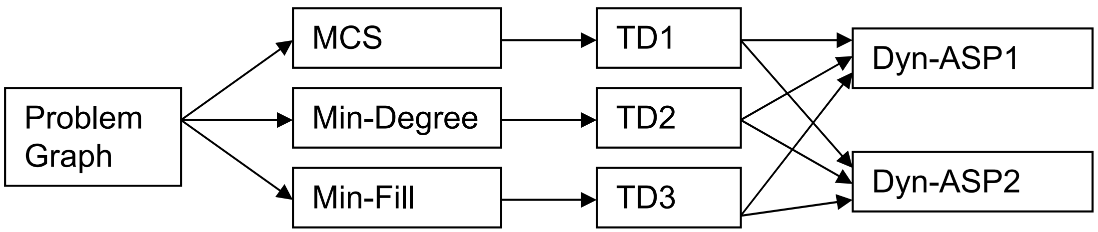
- Hyperheuristics [11]
  - Used to select between different low level heuristics
  - See different approaches used in hyperheuristic competition:

    http://www.asap.cs.nott.ac.uk/chesc2011/
- Dynamic Algorithm selection with reinforcement learning [12]

[11] Burke, E. K., M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu (2010). Hyper-heuristics: A Survey of the State of the Art, School of Computer Science and Information Technology, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747.

[12] Michail G. Lagoudakis, Michael L. Littman: Algorithm Selection using Reinforcement Learning. ICML 2000: 511-518

Algorithm selection for tree-decomposition based algorithms



- Select one of algorithms based on tree decomposition features (tree width, size of tree decomposition, …)
- Classification
  - Predict the algorithm to be used based on features of the input instance
- Regression
  - Predict the running time of both algorithms and select then the more efficient algorithm

Reference:

Michael Morak, Nysret Musliu, Reinhard Pichler, Stefan Rümmele, Stefan Woltran. Evaluating Tree-Decomposition Based Algorithms for Answer Set Programming. *Learning and Intelligent Optimization Conference (LION 6), Paris, Jan 16-20, 2012. Lecture Notes in Computer Science, Volume 7219, pages 130-144, Springer.*

Case Studies

- Case study 1:
  - Application of Machine Learning for Algorithm Selection in Graph Coloring

    References:
    - Martin Schwengerer. Algorithm Selection for the Graph Coloring Problem. *Master Thesis, Vienna University of Technology, 2012.*
    - Nysret Musliu, Martin Schwengerer. Algorithm Selection for the Graph Coloring Problem. *Learning and Intelligent OptimizatioN Conference (LION 7), Catania - Italy, Jan 7-11, 2013. Lecture Notes in Computer Science, to appear.*
- Case study 2:
  - Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning

    References:
    - Michael Abseher, Nysret Musliu, Stefan Woltran. Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning. J. Artif. Intell. Res. 58: 829-858 (2017)
    - Michael Abseher, Frederico Dusberger, Nysret Musliu, Stefan Woltran. Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning. IJCAI 2015: 275-282

Algorithm Selection for the Graph Coloring Problem

Nysret Musliu   Martin Schwengerer

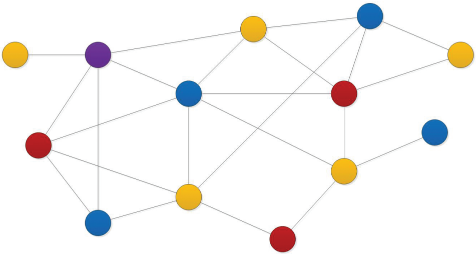
DBAI Group, Institute of Information Systems, Vienna University of Technology

Learning and Intelligent OptimizatioN Conference 2013

Supported by FWF (The Austrian Science Fund) and
FFG (The Austrian Research Promotion Agency).

Graph Coloring

- The Graph Coloring Problem (GCP) is a well-known NP-hard problem.
- Input: Graph $G = (V, E)$
- Objective: assign each node a color such that
  - no adjacent nodes have the same color and
  - the total number of colors k is minimized.



Graph Coloring (cont.)

- Exact approaches are in general only usable up to **100** nodes.
- Several (meta)heuristic approaches:
  - Tabu search
  - Simulated annealing
  - Genetic algorithm
  - Ant colony optimization
  - ...
- **But: None of these techniques is superior to all others.**
- Practical issue: Which heuristic should be used?

Graph Coloring (cont.)

- Exact approaches are in general only usable up to **100** nodes.
- Several (meta)heuristic approaches:
  - Tabu search
  - Simulated annealing
  - Genetic algorithm
  - Ant colony optimization
  - ...
- **But: None of these techniques is superior to all others.**
- Practical issue: Which heuristic should be used?
- Our approach: Select for each instance the algorithm which is expected to give best performance.

Algorithm Selection

- Algorithm Selection Problem by Rice [RICE, 1976]
- Main components:
  - Problem space $\mathtt{P}$
  - Feature space $\mathtt{F}$
  - Algorithm space $\mathtt{A}$
  - Performance space $\mathtt{Y}$
- Task: Find a selector $\mathtt{S}$ that selects for an instance $\boldsymbol{i} \in \boldsymbol{P}$ the best algorithm $\boldsymbol{a} \in \boldsymbol{A}$.

Related Work

- Algorithm selection for other problems
  - SAT (e.g. `SATzilla` [Xu *et al.*, 2008])
  - ASP (e.g. `ME-ASP` [Maratea *et al.*, 2012])
  - TSP (e.g. [Kanda *et al.*, 2011])
  - ...
- Recent research concerning the GCP
  - Predicting performance of DSATUR and TABU search [Smith-Miles *et al.*, 2013]

Graph Coloring using Automated Algorithm Selection

Algorithm selection for the GCP using *machine learning*.

Our system:

- Problem space $\mathtt{P}$: instances of the GCP
- Feature space $\mathtt{F}$: $\mathbf{78}$ different attributes of a graph
- Algorithm space $\mathtt{A}$: state-of-the-art heuristics for the GCP
- Performance criteria $\mathtt{Y}$: lowest $\boldsymbol{k}$ and shortest runtime

As decision procedure $\mathtt{S}$, we use *classification algorithms*.

Features

We identified **78** basic features of a GCP instance that can be calculated in polynomial time based on:

- Graph Size
- Node degree
- Clustering Coefficient
- Clique Size
- Greedy Coloring Algorithms
- Local Search Attributes
- Lower- and upper bounds
- Tree Decomposition

Features

Graph Size Features:
1: **no. of nodes:** n
2: **no. of edges:** m
3,4: **ratio:** $\frac{n}{m}, \frac{m}{n}$
5: **density:** $\frac{2 \cdot m}{n \cdot (n-1)}$

Node Degree:
6-13: **nodes degree statistics:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy

Maximal Clique:
14-20: **normalized by** n: min, max, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy
21: **computation time**
22: **maximum cardinality**

Clustering Coefficient
23: **global clustering coefficient**
24-31: **local clustering coefficient:** min, max, mean, median, $Q_{0.25}, Q_{0.75}$, variation coefficient, entropy
32-39: **weighted local clustering coefficient:** min, max, mean, median, $Q_{0.25}, Q_{0.75}$, variation coefficient, entropy
40: **computation time**

Local Search Probing Features:
41, 42: **avg. impr.:** per iteration, per run
43: **avg no. iterations to local optima (LO)** per a run
44, 45: **no. conflict nodes:** at LO, at end
46, 47: **no. conflict edges:** at LO, at end
48: **no. LO found**
49: **computation time**

Greedy Coloring:
50,51: **no. colors needed:** k_{DSAT}, k_{RLF}
52, 53: **computation time:** t_{DSAT}, t_{RLF}
54, 55: **ratio:** $\frac{k_{DSAT}}{k_{RLF}}, \frac{k_{RLF}}{k_{RLF}}$
56: **best coloring:** $\min(k_{DSAT}, k_{RLF})$
57-72: **independent-set size:** min, max, mean, median, $Q_{0.25}$, $Q_{0.75}$, variation coefficient, entropy

Tree Decomposition:
73: **width of decomposition**
74: **computation time**

Lower- and Upper Bound:
75, 76: **distance:** $\frac{(B_l - B_u)}{B_l}, \frac{(B_u - B_l)}{B_u}$
77, 78: **ratio:** $\frac{B_l}{B_u}, \frac{B_u}{B_l}$

Algorithm Space

We tested **6** state-of-the-art heuristic algorithms:

- Foo-PartialCol (FPC) [Blöchliger and Zufferey, 2008]
- Hybrid Evolutionary Algorithm (HEA) [Galinier and Hao, 1999]
- Iteraded Local Search (ILS) [Chiarandini and Stützle, 2002]
- Multi-Agent Fusion Search (MAFS) [Xie and Liu, 2009]
- MMT [Malaguti *et al.*, 2008]
- TABUCOL (TABU) [Hertz and de Werra, 1987]

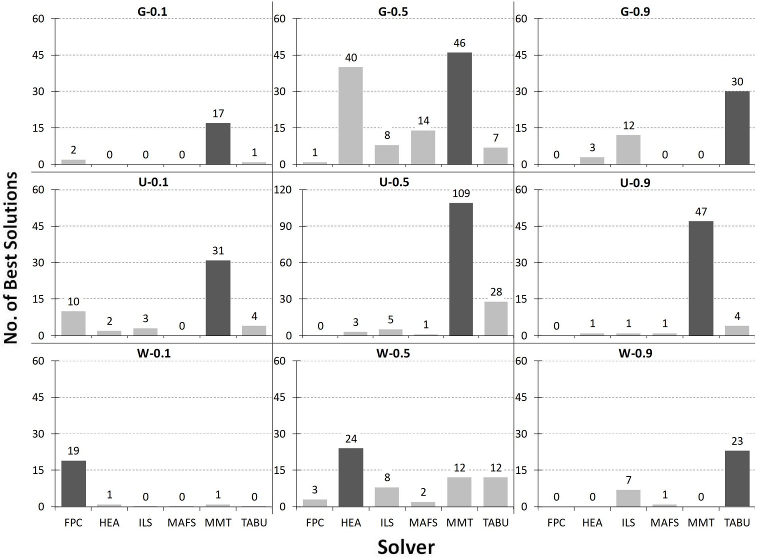
Benchmark Data

- **3** publicly available instance sets:
  - `chi500`: 520 graphs with 500 vertices[1]
  - `chi1000`: 740 graphs with 1000 vertices[1]
  - `dimacs`: 174 graphs of the DIMACS challenge[2]

- Each instance is tested **10** times.
- Total runtime: roughly **90.000** CPU hours.
- Focus on hard instances (**859** of the **1265** graphs).

[1] available at `http://www.imada.sdu.dk/~marco/gcp-study/`
[2] available at `http://mat.gsia.cmu.edu/COLOR04/`

Solver Performance



Number of hard instances from the set `chi1000` on which the algorithms show best performance.

Selection Procedure

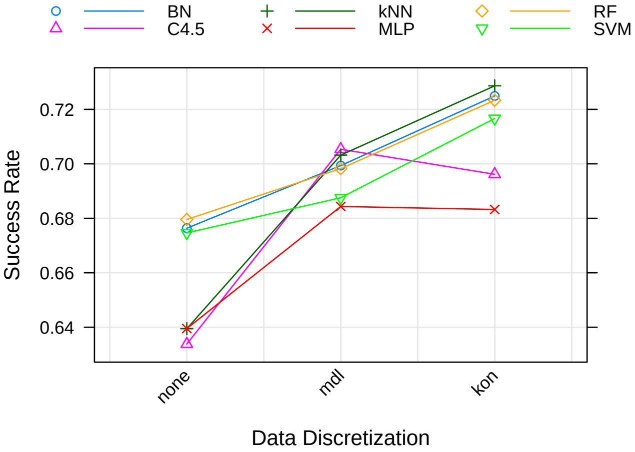
- We tested *6* popular classification algorithms:
  - Bayesian Networks (BN)
  - C4.5 Decision Trees (DT)
  - k-Nearest Neighbor (kNN)
  - Random Forests (RF)
  - Multilayer Perceptrons (MLP)
  - Support-Vector Machines (SVM)
- with several parameter configurations for each classifier.

Other Important Issues

In addition, we experimented with:

- Effect of Data Preparation:
  - Study the effect of two *discretization methods*:
    - The classical minimum-descriptive length (MDL) and
    - Kononenko's criteria (KON).
- Feature Selection:
  - Use *best-first* and a *genetic search* strategy to identify useful features.

Effect of Discretization



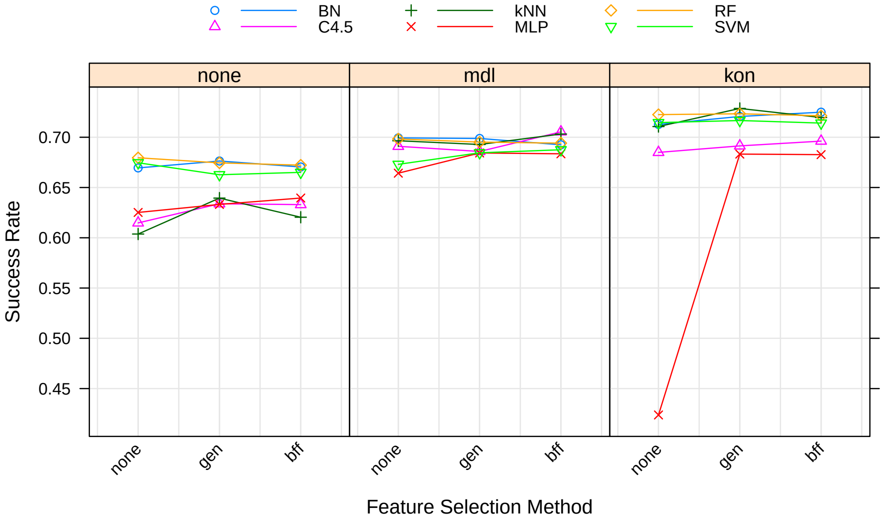
- Discretization improves the performance of almost any classifier.
- KON is slightly better than MDL for some classifiers.

Feature Selection (cont.)

Starting with our *78* basic attributes, we:

1. Apply *best-first* and a *genetic search* strategy to identify two subsets U_b and U_g.
2. Add the product $x_i \cdot x_j$ and the quotient x_i / x_j of each pair of features $x_i, x_j \in (U_b \cup U_g)$ as additional features.
3. Apply again *best-first* and a *genetic search*.

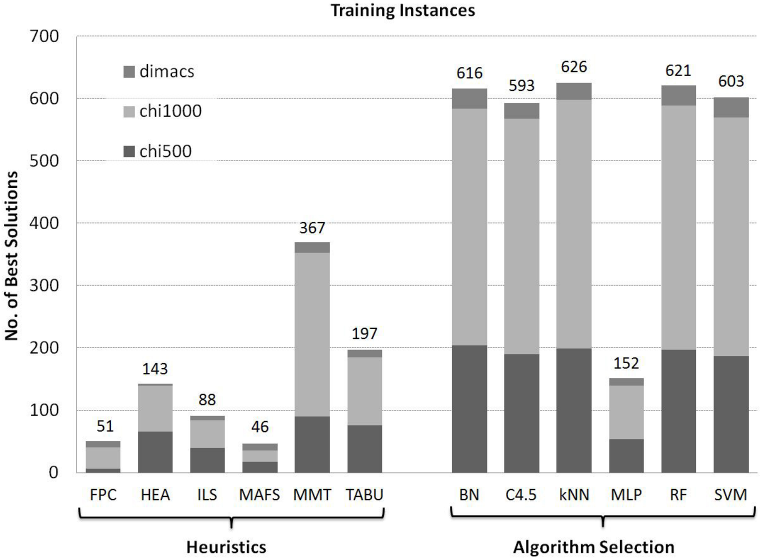
Results of Feature Selection



Results of Feature Selection and Data Discretization

- Use the feature subset obtained by the *genetic search*.
- Data discretized with *Kononenko's criteria*.

Results on the Training Data



Results of **20** runs of a *10-fold cross-validation* using *KON* and the results of the *genetic search*.

Results on the Training Data (cont.)

- We further applied a *corrected resampled T-test* with $\alpha = \mathbf{0.05}$ using cross-validation.

Results:

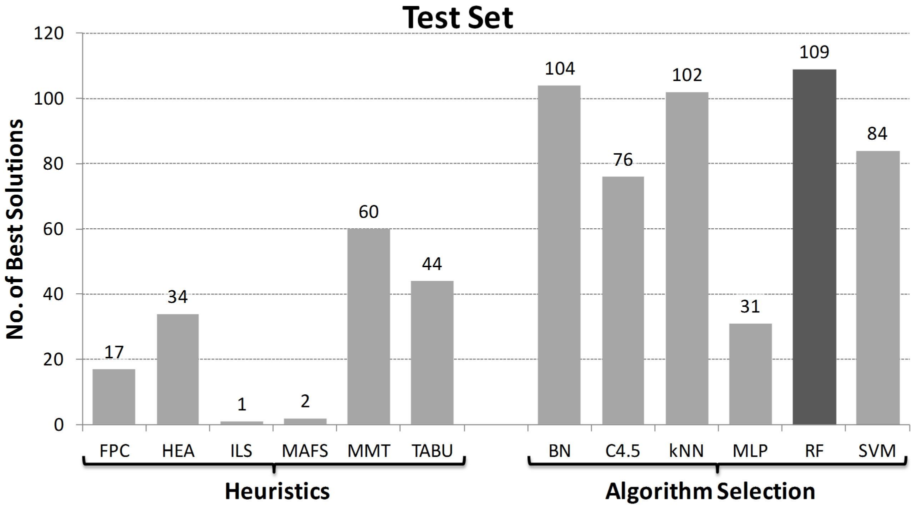
- *BN*, *kNN* and *RF* are significant better than *DT*.
- All other pairwise comparisons do not show significant differences.

Evaluation on the Test Set

- We create a *test set* with **180** graphs of different class, size and density.

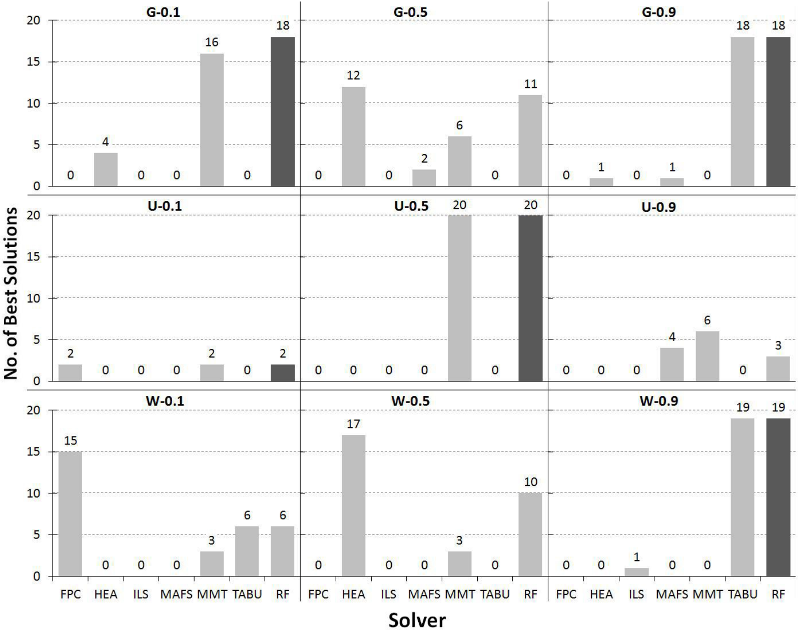
- Our system based on automated algorithm selection:
  - Using the all **6** *heuristics*.
  - Trained with the *benchmark data*.
  - Data discretized with *Kononenko's criteria*.

Evaluation on the Test Set - Results



Number of best solutions per solver. The dark bar denotes the approach that shows on the highest number of instances the best performance.

Evaluation on the Test Set (cont.)



Conclusion

- We applied automated algorithm selection for the GCP.
- Key features:
  - *78* basic features of an GCP instance.
  - *6* state-of-the-art heuristics.
  - Training data of **859** hard graphs.
  - *Classification algorithms* as selection procedure.

Results:

- Classification algorithms predicts for up to $\mathbf{70.39\%}$ of the graphs the most suited algorithm.
- Improvement of $\mathbf{+33.55\%}$ compared with the best solver.

References I

- Blöchliger, I. and Zufferey, N. (2008).
  Computers & Operations Research **35**, 960–975.
- Chiarandini, M. and Stützle, T. (2002).
  An application of Iterated Local Search to Graph Coloring.
  In Johnson, D. S., Mehrotra, A., and Trick, M. A., editors, *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pages 112–125, Ithaca, New York, USA.
- Galinier, P. and Hao, J.-K. (1999).
  Journal of Combinatorial Optimization **3**, 379–397.
- Hertz, A. and de Werra, D. (1987).
  Computing **39**, 345–351.
- Kanda, J., Carvalho, A., Hruschka, E., and Soares, C. (2011).
  Neural Networks **8**.
- Malaguti, E., Monaci, M., and Toth, P. (2008).
  INFORMS Journal on Computing **20**, 302–316.
- Maratea, M., Pulina, L., and Ricca, F. (2012).
  Applying Machine Learning Techniques to ASP Solving.
  In Dovier, A. and Costa, V. S., editors, *ICLP (Technical Communications)*, volume 17 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.
- Rice, J. R. (1976).
  Advances in Computers **15**, 65–118.
- Smith-Miles, K., Wreford, B., Lopes, L., and Insani, N. (2013).
  Predicting Metaheuristic Performance on Graph Coloring Problems Using Data Mining.
  In *Hybrid Metaheuristics*, Studies in Computational Intelligence, pages 417–432.
- Xie, X.-F. and Liu, J. (2009).
  Journal of Combinatorial Optimization **18**, 99–123.

References II

- XU, L., HUTTER, F., HOOS, H. H., and LEYTON-BROWN, K. (2008).
  Journal of Artificial Intelligence Research **32**.

Appendix - Evaluation on the Test Set (cont.)

Solver	No. Best Solution	$s(c, I, A)$ (%)	$err(k, i)$ (%)	Rank avg	Rank σ
		Heuristics (H)			
FPC	17	11.18	25.42	3.29	1.42
HEA	34	22.37	14.91	2.66	1.38
ILS	1	0.66	21.73	3.82	1.36
MAFS	2	1.32	30.17	4.62	1.52
MMT	60	39.47	**3.78**	2.76	1.84
TABU	44	28.95	19.23	2.58	1.29
		Algorithm Selection (AS)			
BN	104	68.42	5.16	1.59	1.08
C4.5	76	50.00	5.86	2.21	1.50
kNN	102	67.11	3.82	1.52	0.91
MLP	31	20.39	24.90	3.14	1.66
RF	**109**	**71.71**	5.44	**1.41**	**0.78**
SVM	84	55.26	8.32	1.97	1.38
Best (H)	60	39.47	**3.78**	2.58	1.29
Best (AS)	**109**	**71.71**	3.82	**1.41**	**0.78**

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

dbai
Database and Artificial Intelligence Group

Improving the Efficiency of Dynamic Programming on Tree Decompositions via Machine Learning

Michael Abseher, Frederico Dusberger, Nysret Musliu, Stefan Woltran
TU Wien

Der Wissenschaftsfonds.

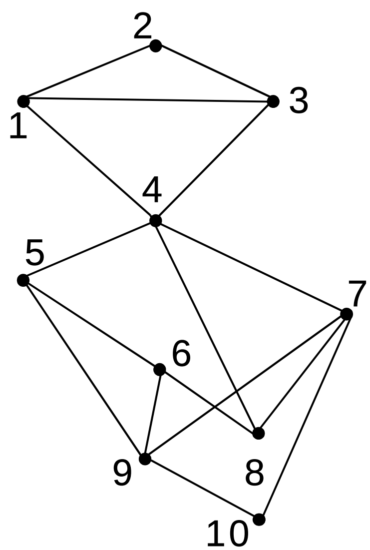
The work is supported by the Austrian Science Fund

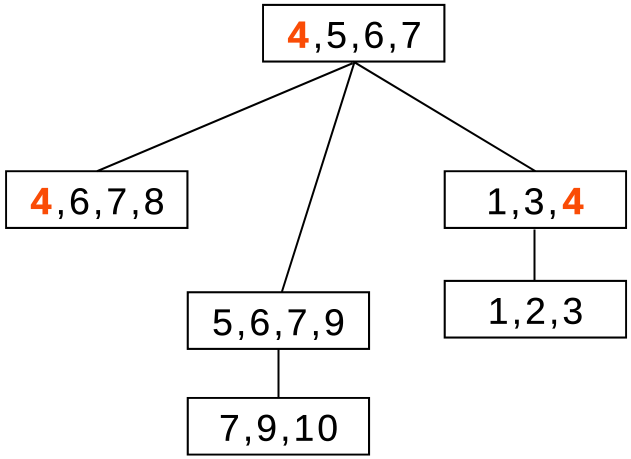
IJCAI 2015

Introduction

- Many NP-hard problems are known to become tractable for instances whose treewidth is bounded by some constant *k*
- A promising approach for solving problems using tree decompositions:
  - Compute a tree decomposition with small width
  - Compute the solutions by a dynamic programming algorithm that consecutively solves the respective sub-problems

Tree decomposition of a graph





All pairs of connected vertices appear in some node of the tree

Connectedness condition for *vertices*

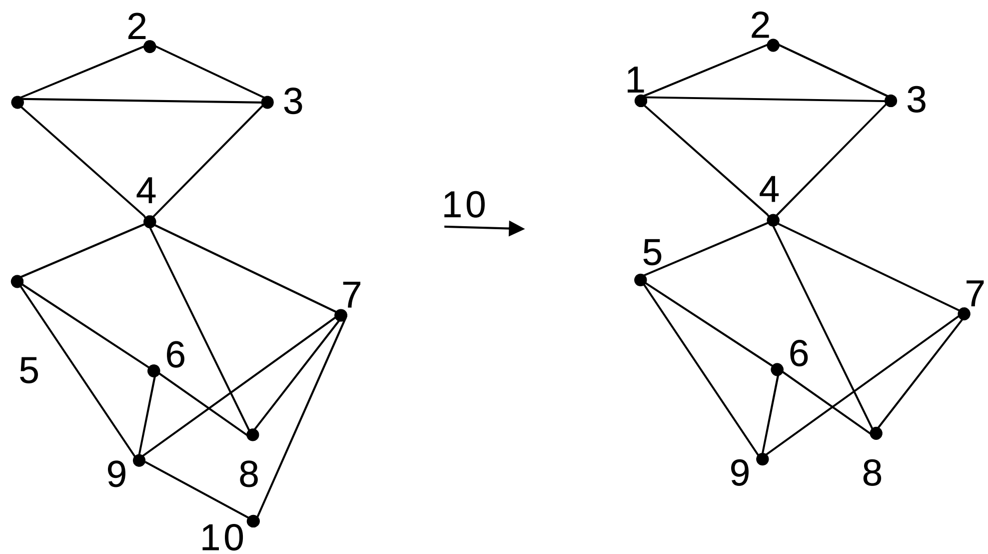
Width: (*number of vertices in the largest tree node) -1 = 3*

Treewidth: *minimal width over all possible tree decompositions*

Generating tree decompositions

- For the given problem find the tree decomposition with minimal width -> NP hard
- There exist perfect elimination ordering which produces tree decomposition with treewidth (smallest width)
- Tree decomposition problem → search for the best elimination ordering of vertices!

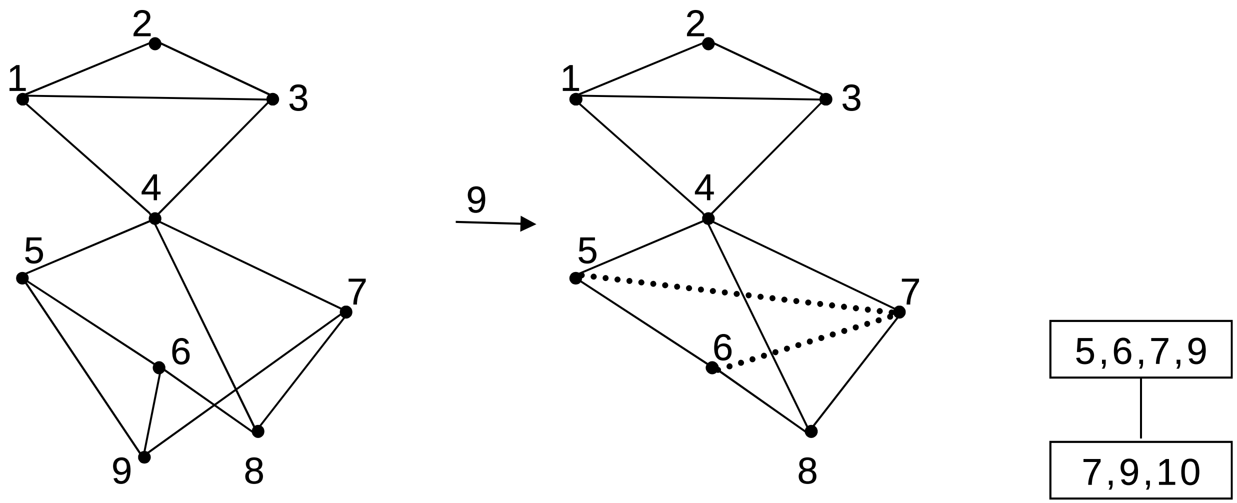
Perfect Elimination Ordering



7,9,10

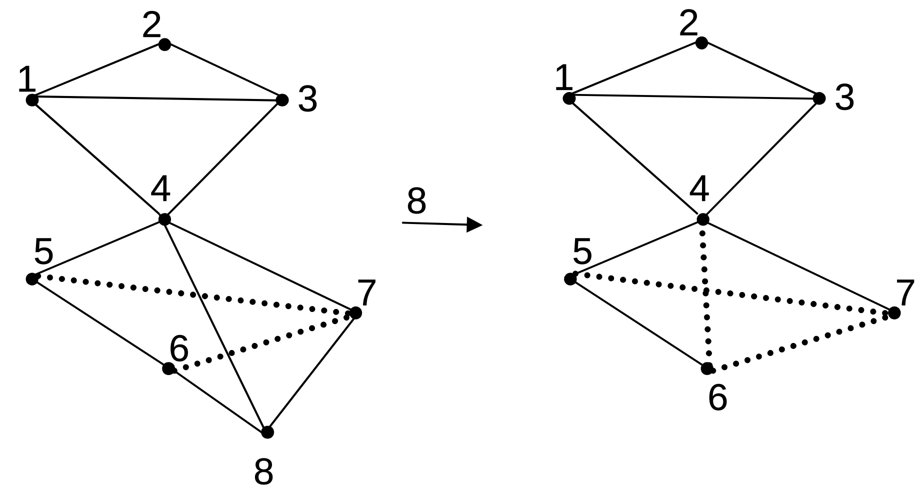
Vertex 10 is eliminated from the graph. All neighbors of 10 are connected and a tree node is created that contains vertex 10 and its neighbors

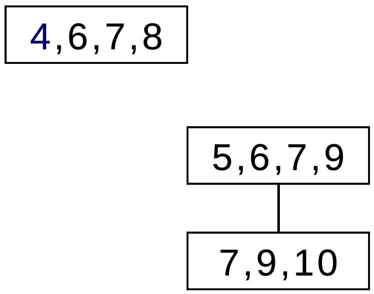
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



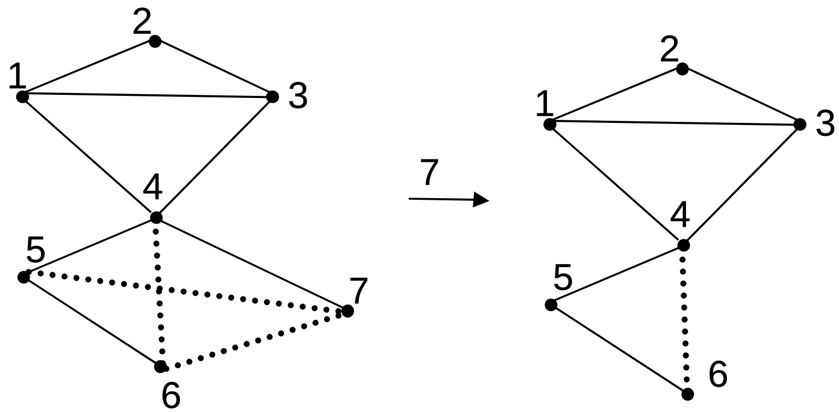
Vertex 9 is eliminated from the graph. All neighbors of vertex 9 are connected and a new tree node is created

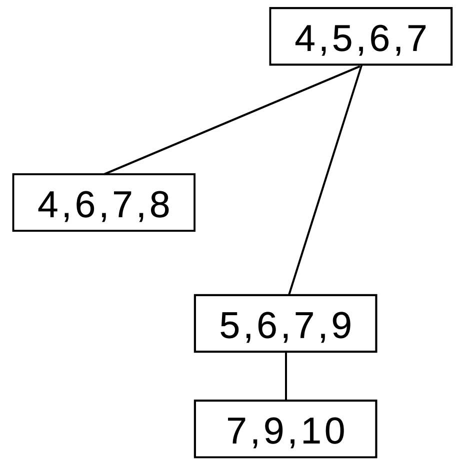
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



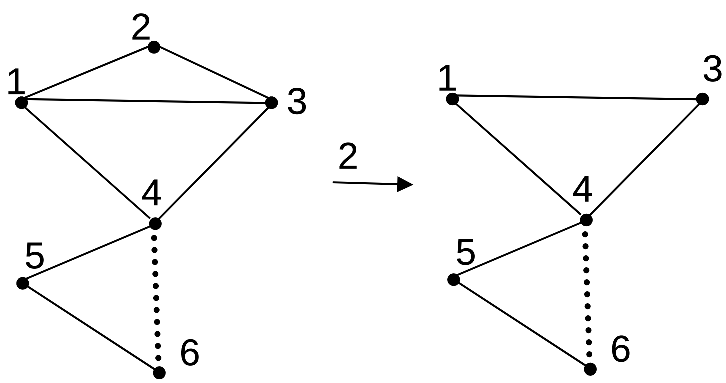


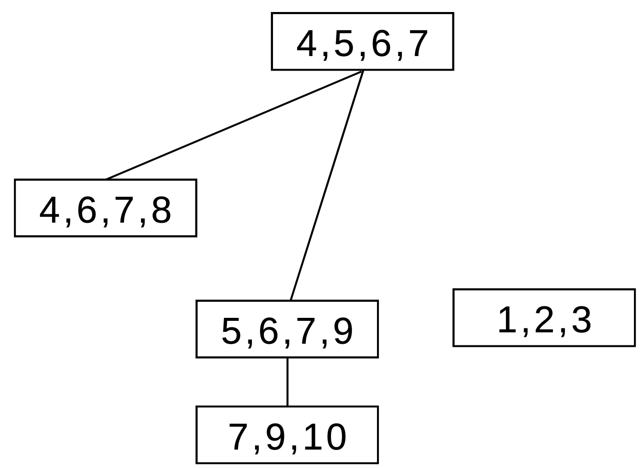
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



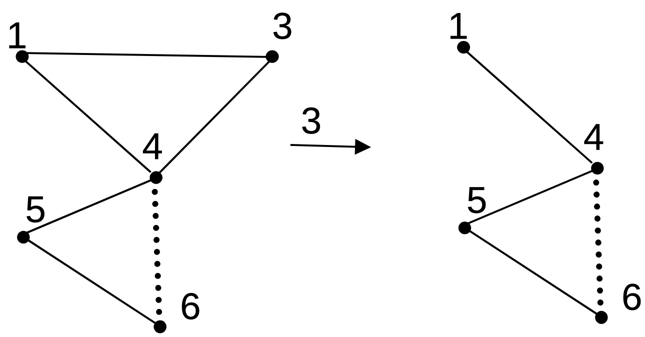


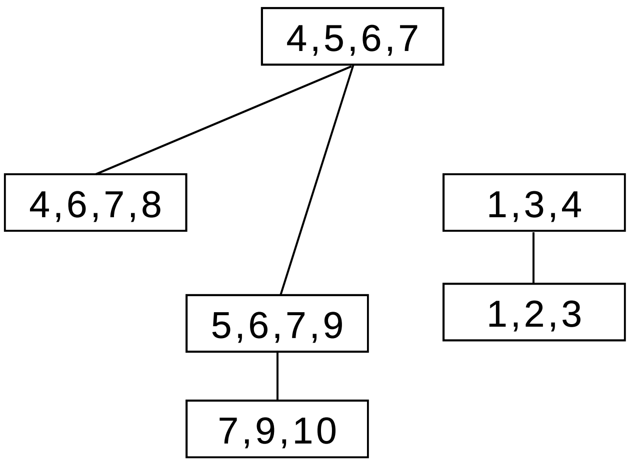
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



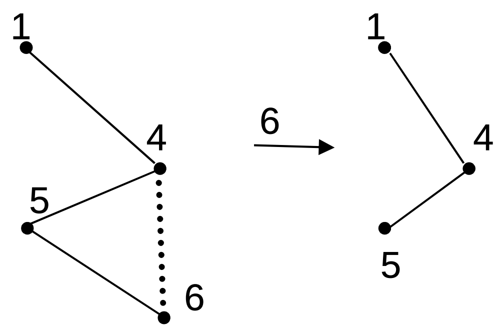


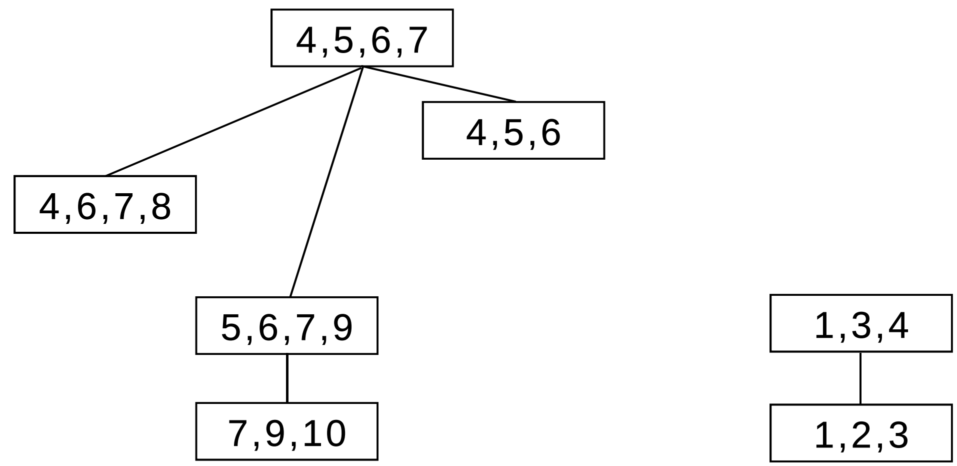
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



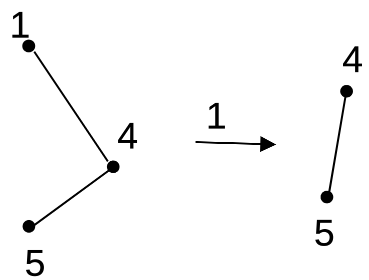


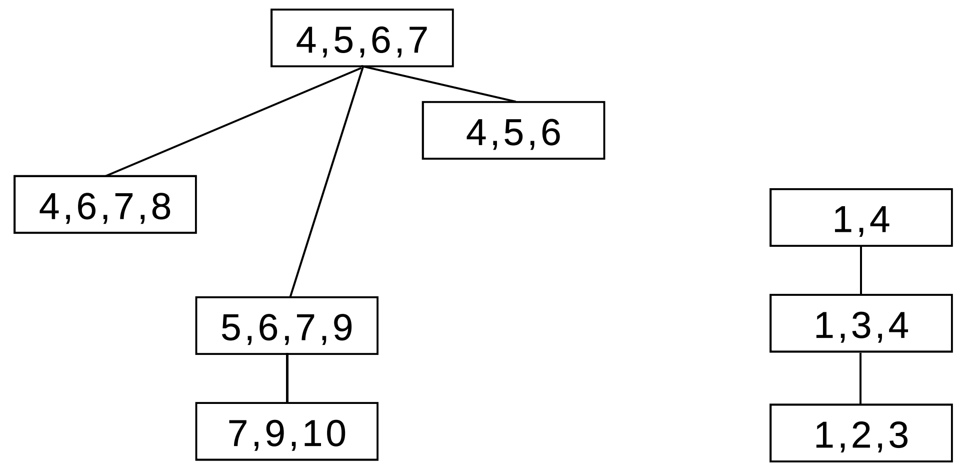
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



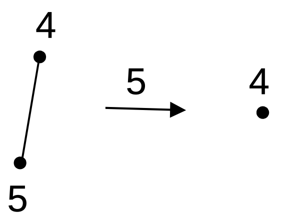


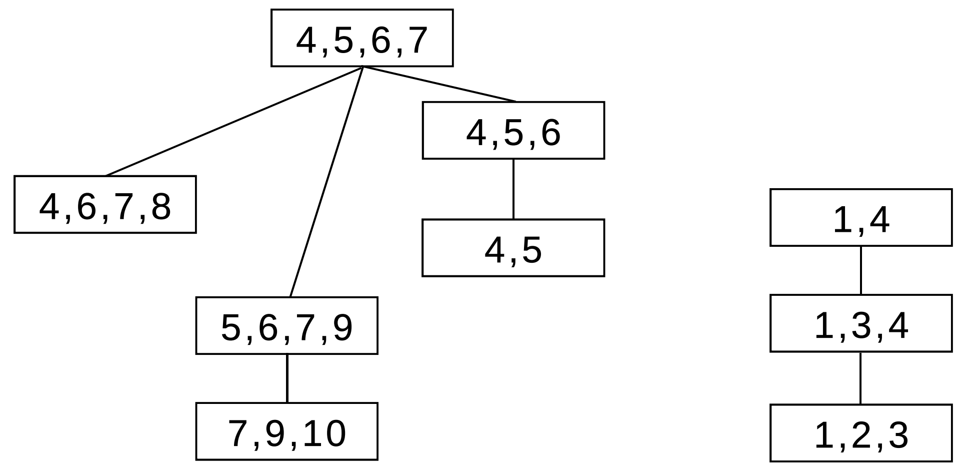
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



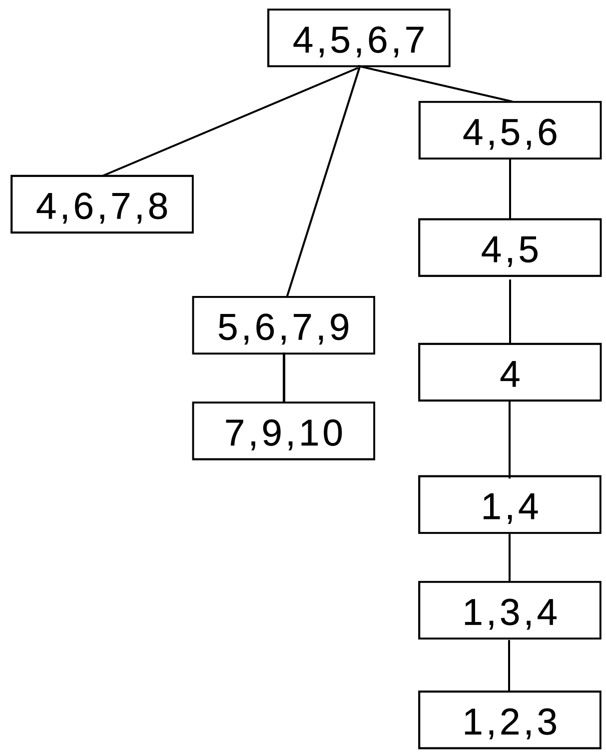


Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

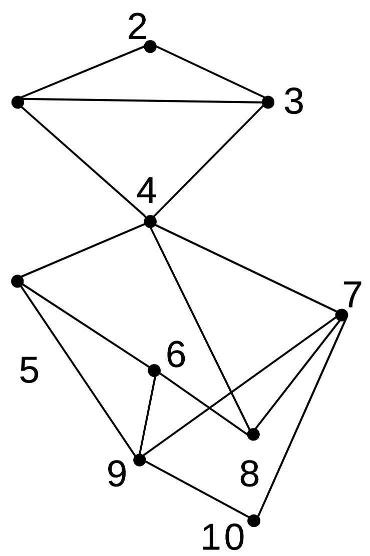


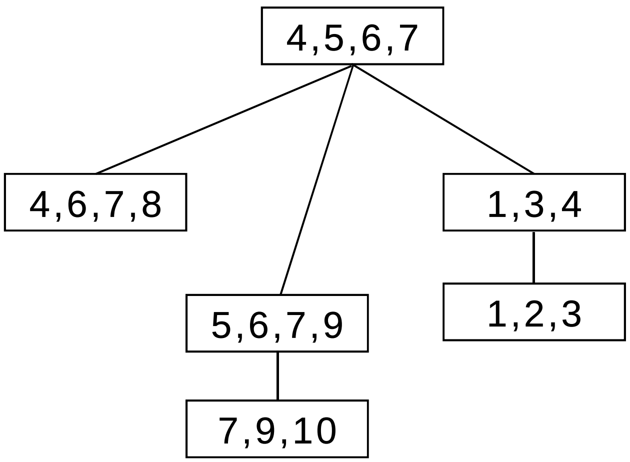


Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4



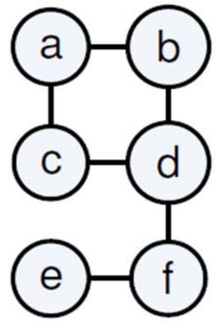


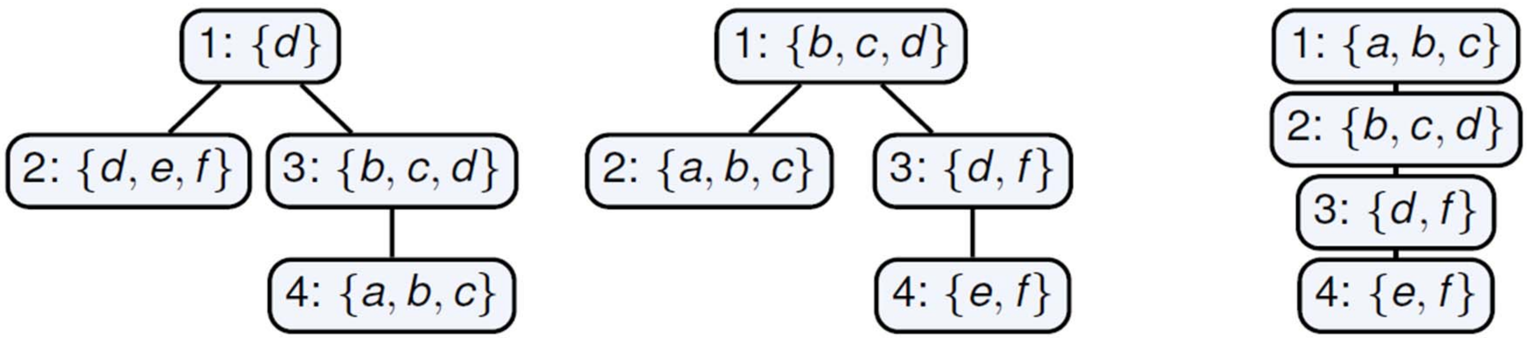
Elimination ordering: 10, 9, 8, 7, 2, 3, 6, 1, 5, 4

Algorithms for tree decompositions

- Exact Methods
  - Branch and bound algorithms
  - A* algorithm
- Greedy methods
  - Maximum Cardinality Search (MCS)
  - Min-fill
  - Min-degree
- Metaheuristic methods
  - Tabu Search
  - Genetic/Memetic Algorithms
  - Iterated Local Search
  - Ant Colony Optimization

- A problem instance has various tree decompositions:





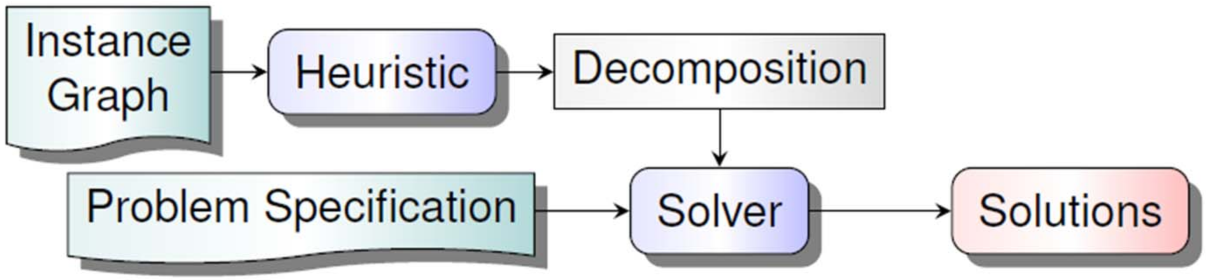
Observation

- Experiments show that the width is likely not the only important parameter having influence on the runtime of dynamic programming algorithms

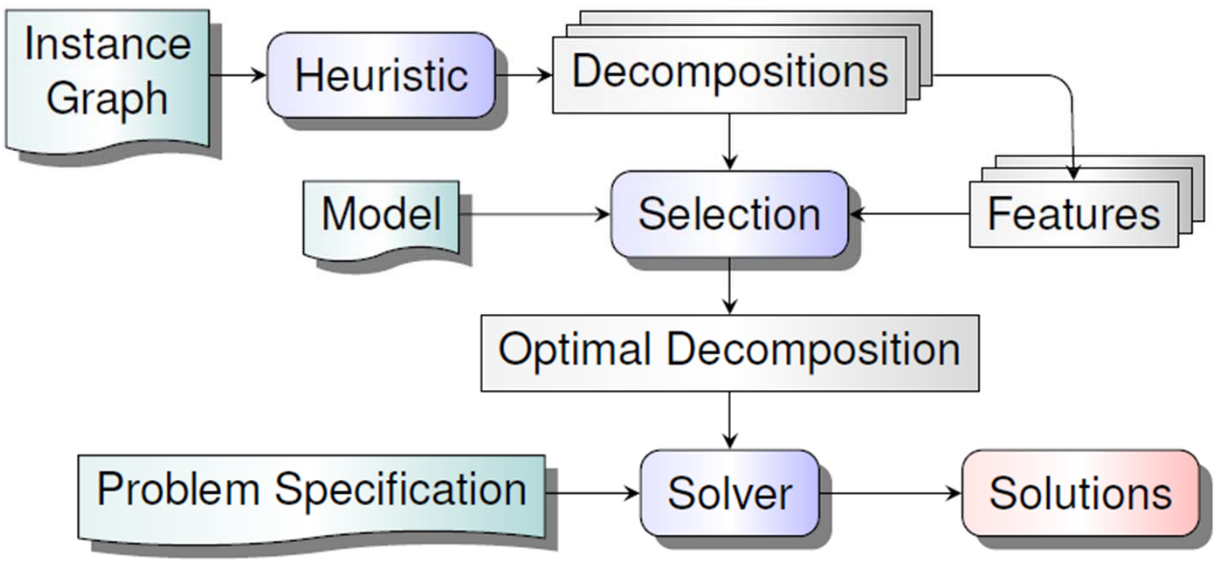
- Even decompositions of the same width often yield extremely diverging runtimes

- How to determine the decomposition which promises best performance?

Improving the efficiency via machine learning

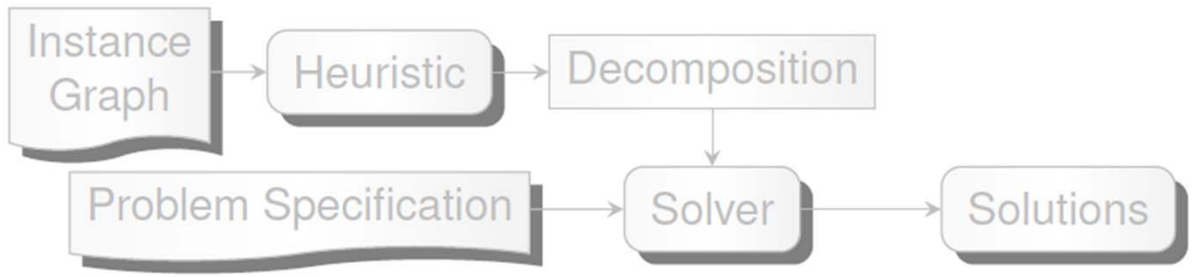


(a) Standard Approach

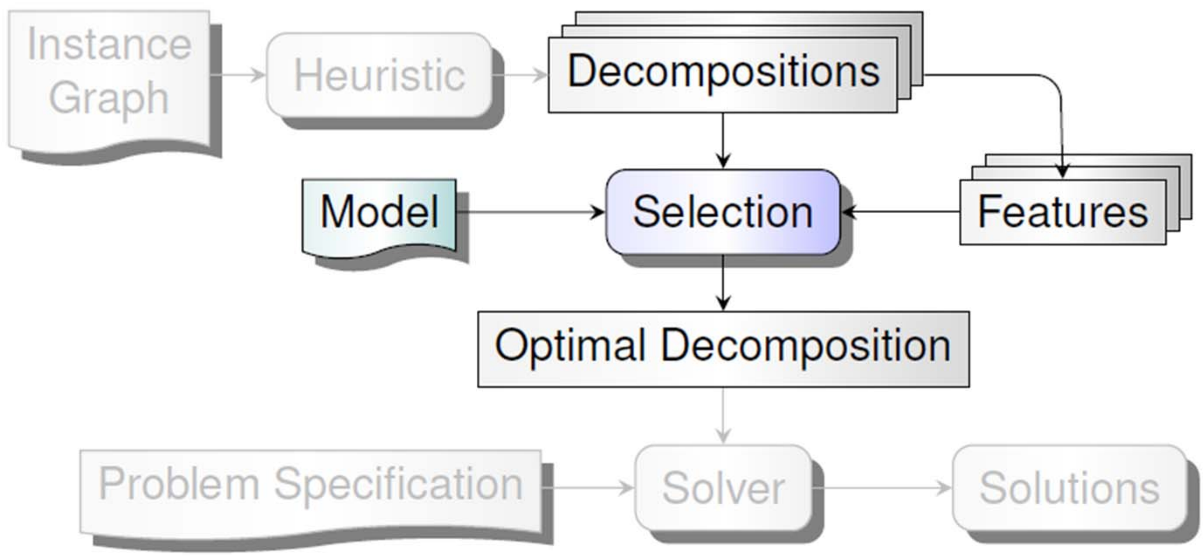


(b) Improved Approach

Selection of tree decomposition



(a) Standard Approach



(b) Improved Approach

Features of tree decomposition

Decomposition Size:

- BagSize*
- Σ BagSize
- NodeCount
- ContainerCount*

Node Features•:

- Depth*
- BagSize*
- NodeCount
- Percentage

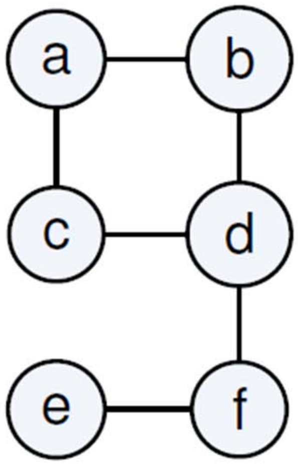
Structural Features:

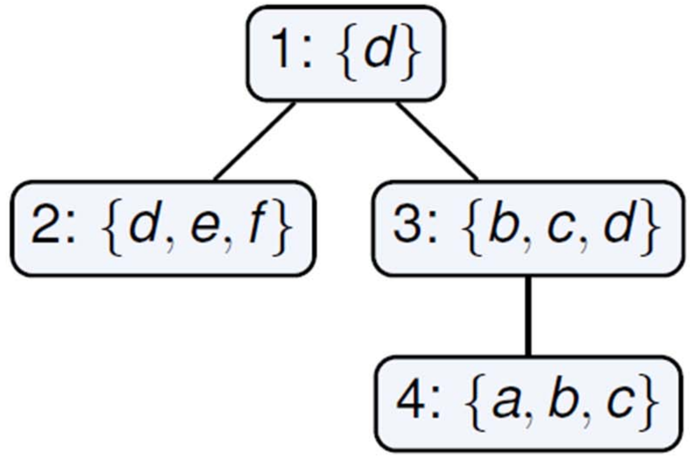
- ItemLifetime*
- JoinNodeDistance*
- NumberOfChildren*
- BalancednessFactor
- AdjacencyRatio*
- ConnectednessRatio*
- NeighborCoverageRatio*

* Mean, Standard Deviation, Median, Minimum, Maximum

• Separately for Introduce-Node, Forget-Node, Join-Node, Leaf-Node

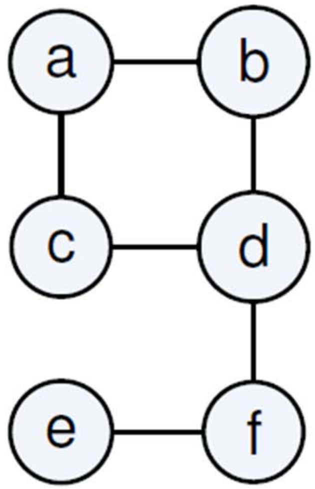
Feature BagSize

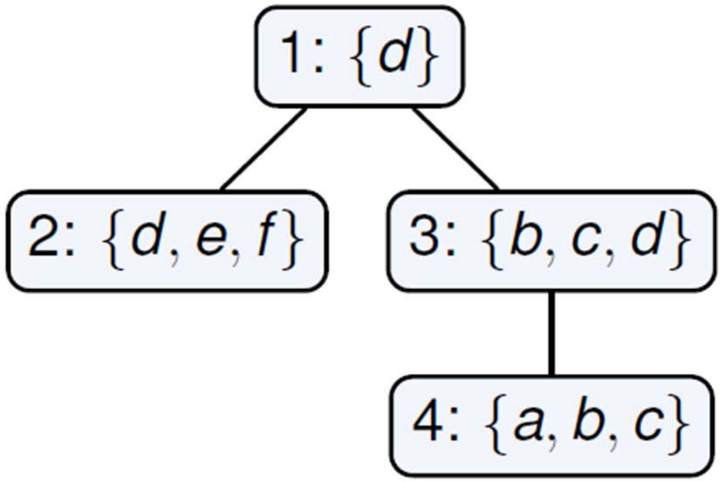




Node	BagSize
1	1
2	3
3	3
4	3
Total:	**10**

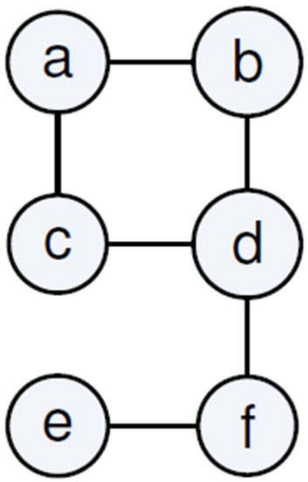
Feature ContainerCount

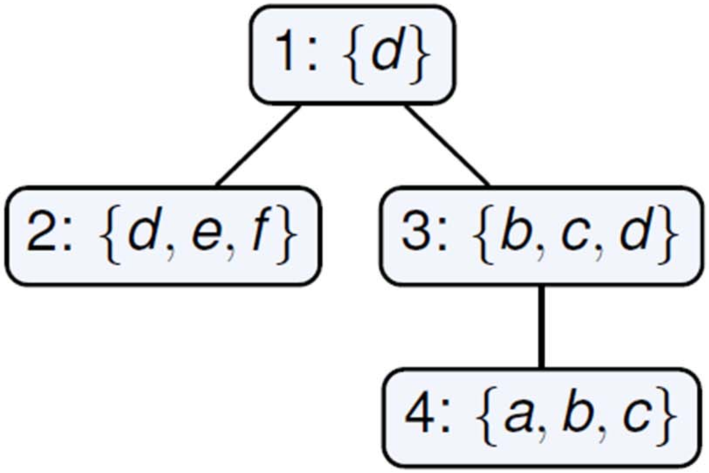




Vertex	Containers
a	1
b	2
c	2
d	3
e	1
f	1

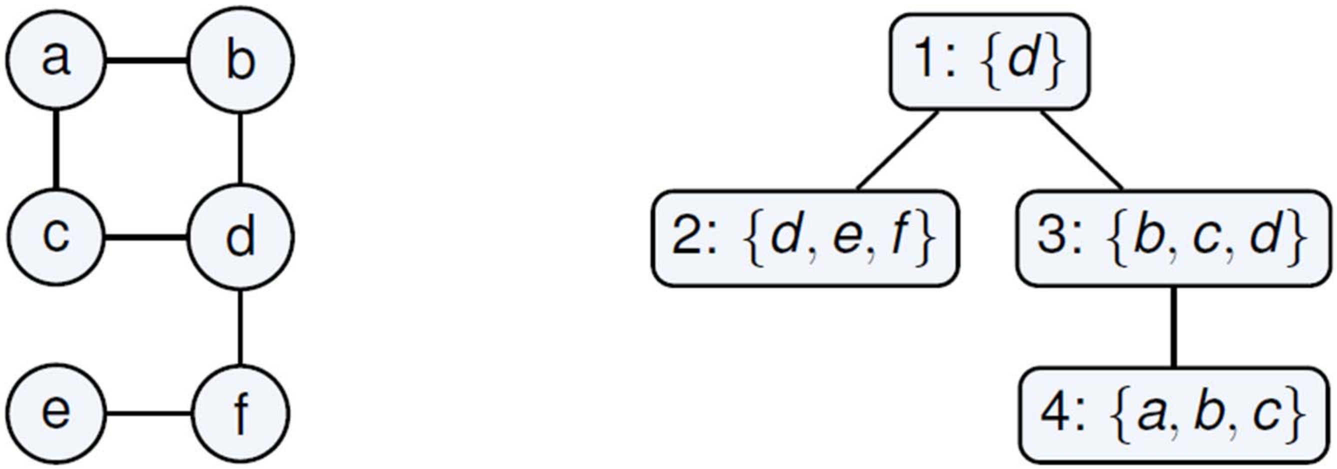
Feature ItemLifetime





Vertex	Lifetime
a	1
b	2
c	2
d	2
e	1
f	1

Feature AdjecencyRatio

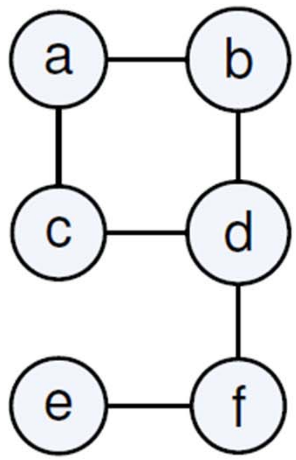


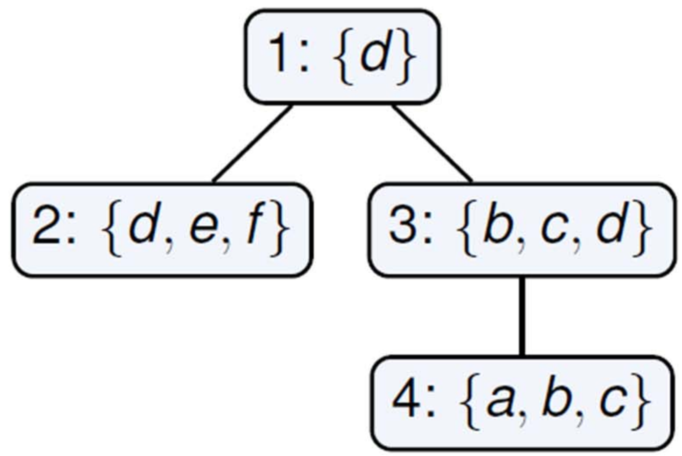
AdjacencyRatio of a Bag $\chi(i)$

$$AdjacencyRatio(\chi(i)) = \sum_{x \in \chi(i)} |N(x) \cap \chi(i)| / max(1, |\chi(i)|)$$

$$\text{where } N(x) = \{y \in V : (x, y) \in E, x \neq y\}$$

Feature NeighborCoverageRatio





NeighborCoverageRatio of a Bag $\chi(i)$

$$NeighborCoverageRatio(\chi(i)) = \sum_{x \in \chi(i)} \frac{|N(x) \cap \chi(i)|}{|N(x)|} / max(1, |\chi(i)|)$$

$$\text{where } N(x) = \{y \in V : (x, y) \in E, x \neq y\}$$

Methodology

- Algorithm Space:
  - D-FLAT (D):

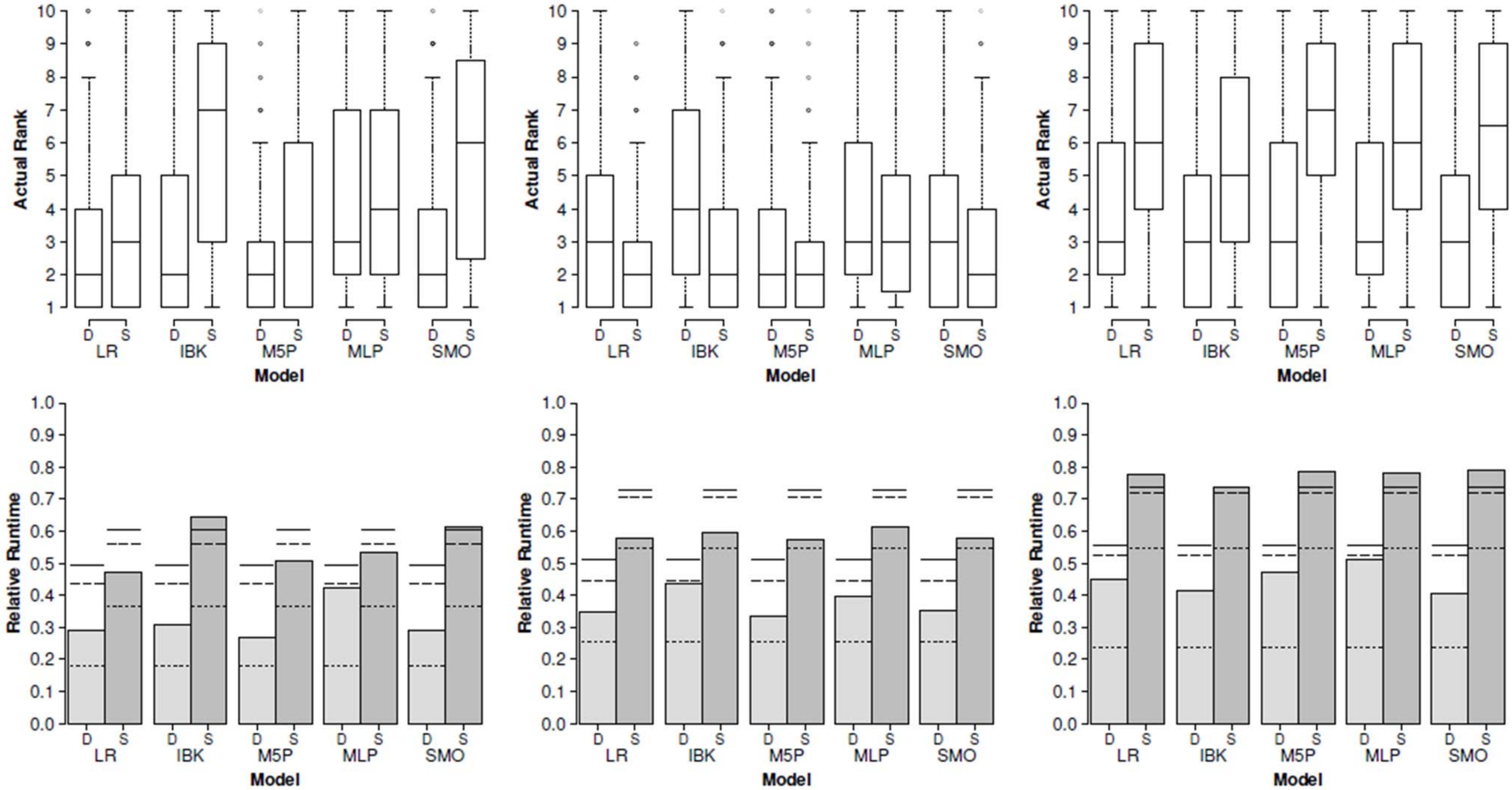
    Based on Answer Set Programming
  - SEQUOIA (S):

    Based on a solver for Monadic Second-Order Logic
- Problem Space:
  - 3-Colorability, Minimum Dominating Set, Connected Vertex Cover
  - Graphs based on real-world instances
- Feature Space:
  - Width, NodeCount, ContainerCount
  - … and more than 70 additional features

Methodology

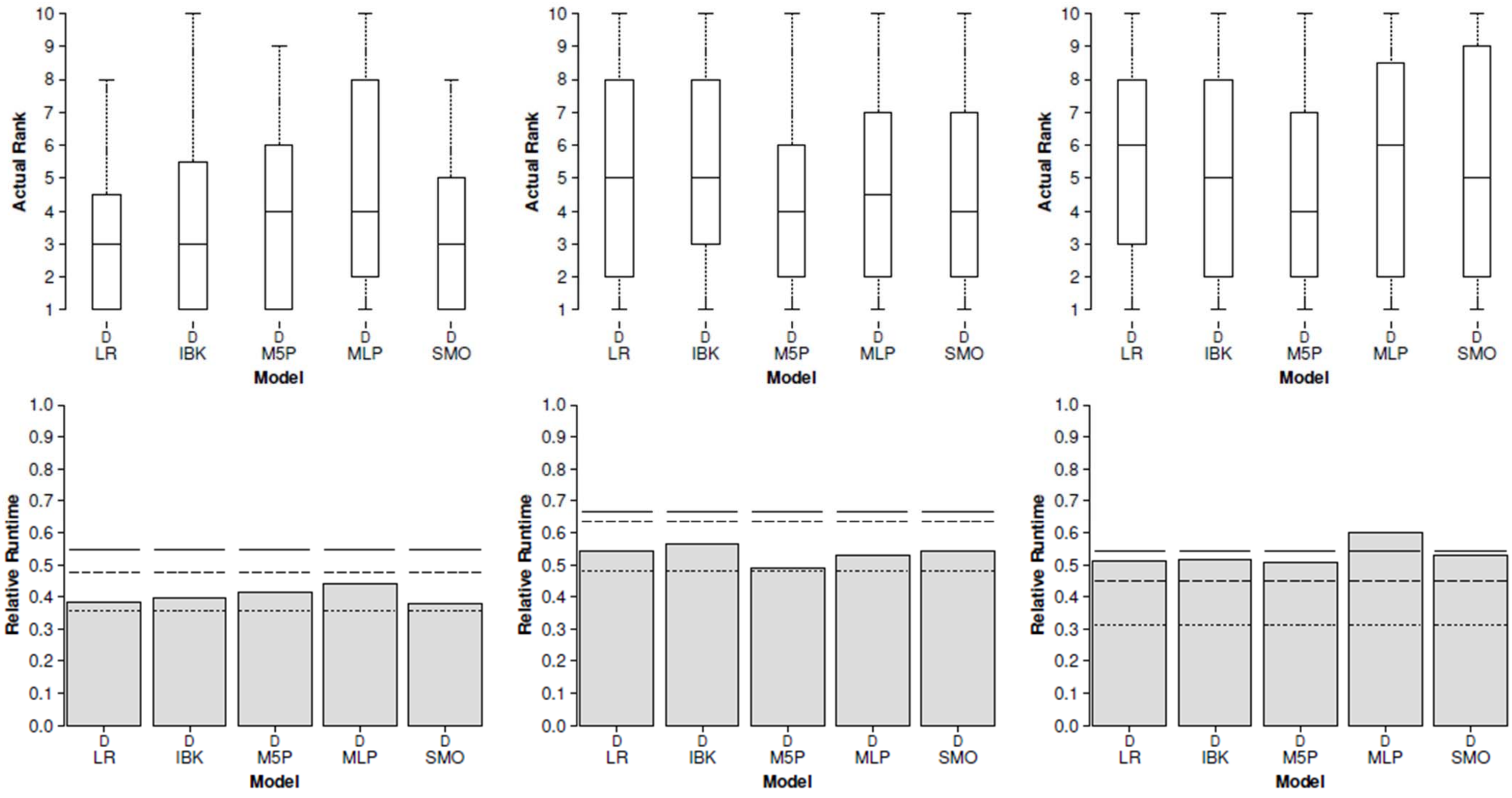
- Training data:
  - 900 tree decompositions for each problem and solver
  - New features
  - Runtimes of dynamic programming algorithm
- Machine learning techniques:
  - Linear Regression (LR)
  - k-Nearest Neighbor (IBK)
  - M5P Regression Tree (M5P)
  - Multi-Layer Perceptron (MLP)
  - Support-Vector Machines (SMO)

Results: Random Instances



(a) 3-Colorability (b) Minimum Dominating Set (c) Connected Vertex Cover

Results: Real-world Instances



Conclusion and Future Work

- The *width* of a tree decomposition is not a reliable measure to predict the runtime of a dynamic programming algorithm in a real-world setting on its own
- We determined various features of tree decompositions. Our experiments indicate that ...
  - the new features indeed help to find good decompositions
  - computing the novel features is computationally cheap
  - selecting a good decomposition is of negligible effort
  - our concept also pays off in real-world settings
- Future work:
  - Investigation of additional problem domains
  - Identification of the most crucial features
  - Ultimate goal:
    - Development of new heuristics for tree decompositions
    - ... which consider the most important features