Combining Metaheuristics based on Solution Construction with Exact Techniques

Christian Blum

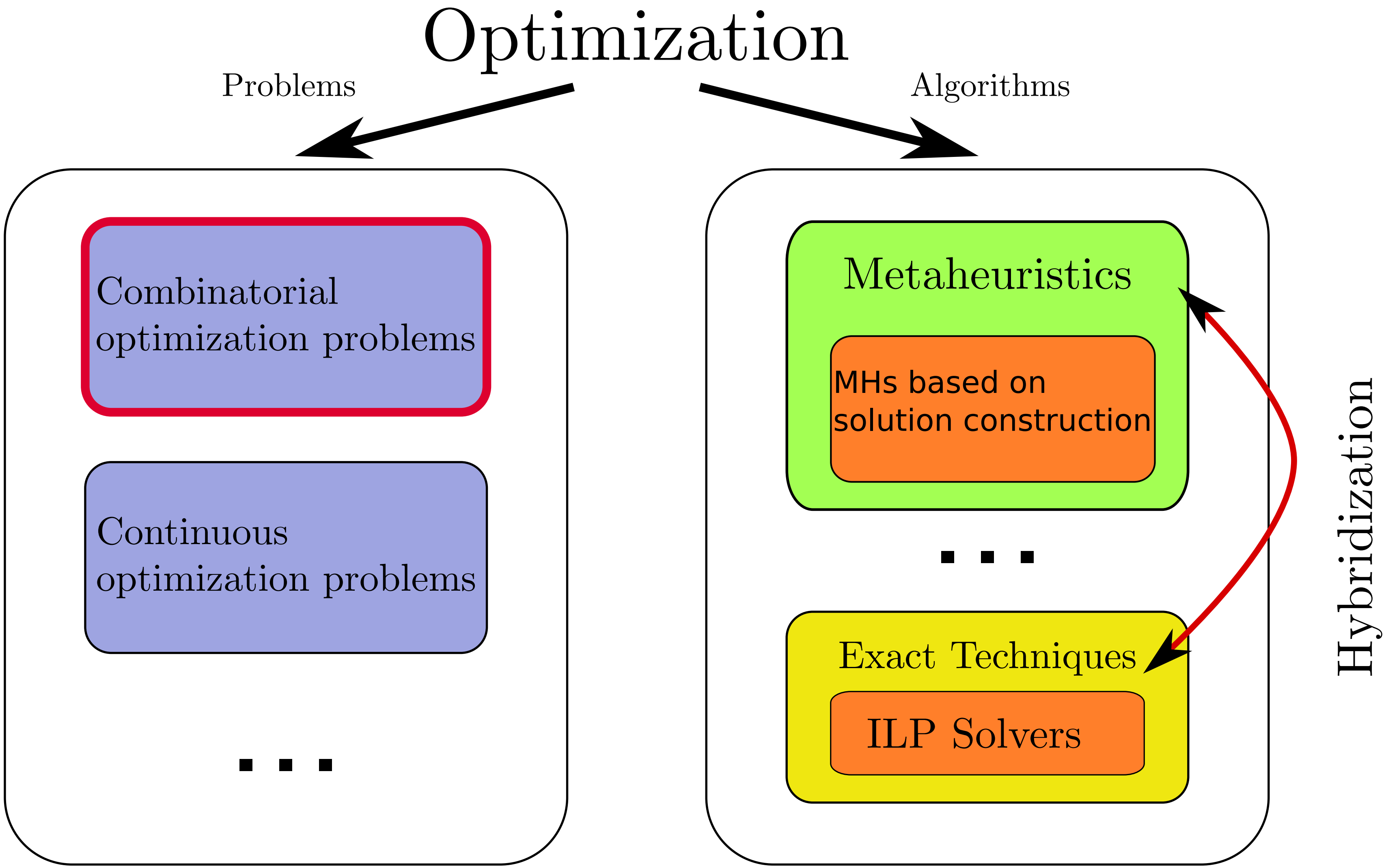
University Of The Basque Country

Ikerbasque, Basque Foundation For Science

ikerbasque
Basque Foundation for Science

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Preliminaries: Preparing the Grounds



Outline

- **Hybrid Metaheuristics**
- **Approach 1:** **Beam-ACO** (2005)
- **Approach 2:** **Construct, Merge, Solve & Adapt (CMSA)** (2015)
- **Application:** **Repetition-free Longest Common Subsequence**
- **Relation:** **CMSA with Large Neighborhood Search**
- **Conclusions / Future Work**

Hybrid metaheuristics: definition

Definition: What is a hybrid metaheuristic?

- Problem: a precise definition is not possible/desirable

Possible characterization:

A technique that results from the combination of a metaheuristic with other techniques for optimization

What is meant by: other techniques for optimiation ?

- Metaheuristics
- Branch & bound
- Dynamic programming
- Integer Linear Programming (ILP) techniques

Hybrid metaheuristics: history

History:

- For a long time the different communities co-existed quite isolated
- Hybrid approaches were developed already early, but only sporadically
- Only since about 15 years the published body of research grows significantly:
  1. 1999: CP-AI-OR Conferences/Workshops
  2. 2004: Workshop series on Hybrid Metaheuristics (HM 200X)
  3. 2006: Matheuristics Workshops

Consequence: The term hybrid metaheuristics identifies a new line of research

Motivation behind my work on hybrid metaheuristics

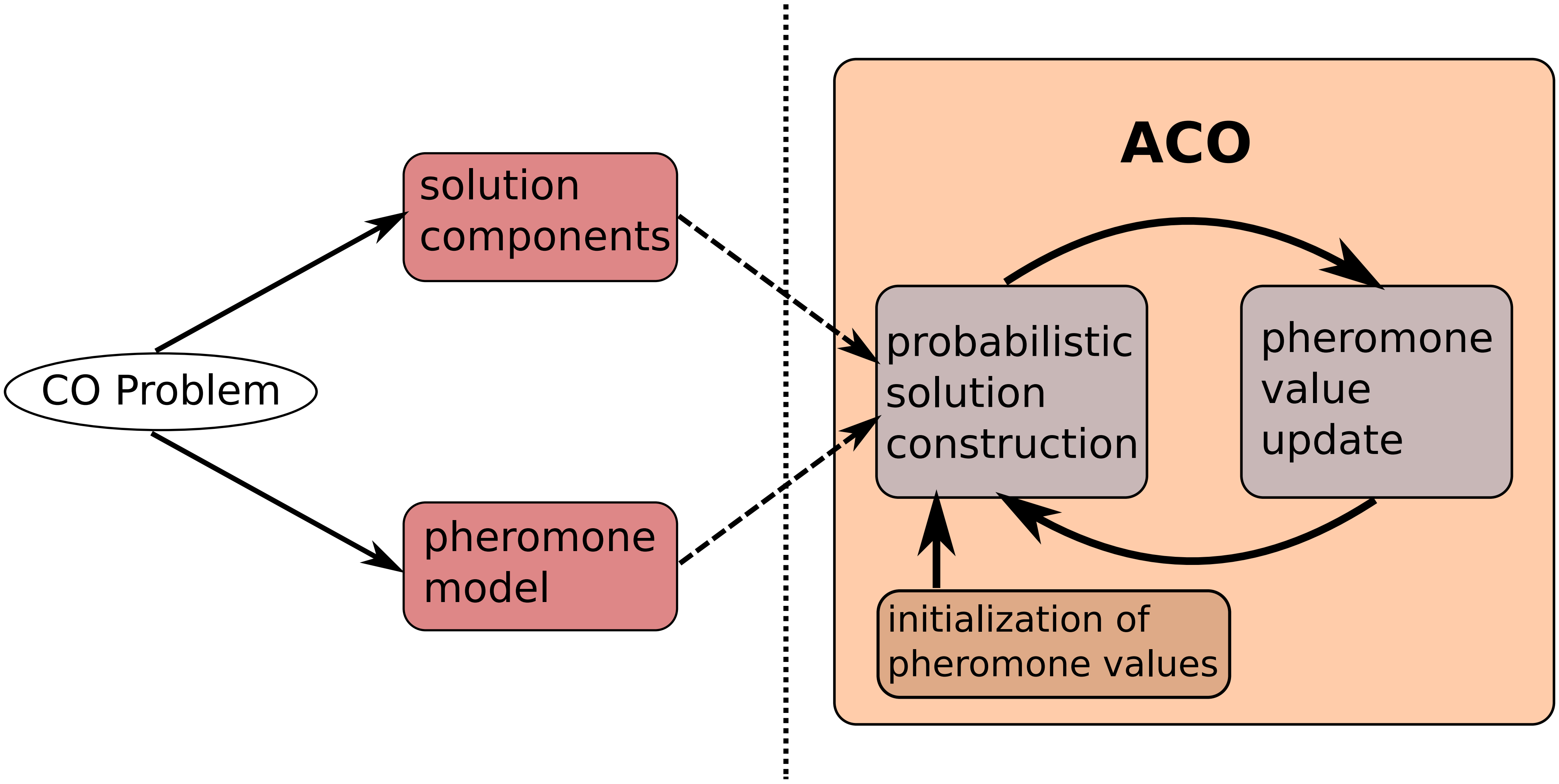
- In the field of metaheuristics we have rules of thumb:
  1. If, for your problem, there is a **good greedy heuristic** apply GRASP or Iterated Greedy
  2. If, for your problem, there is an **efficient neighborhood** apply Iterated Local Search or Tabu Search

- In contrast, for hybrid metaheuristics not much is known
  - ⋆ We only have very few generally applicable techniques
  - ⋆ We do not really know for which type of problem they work well

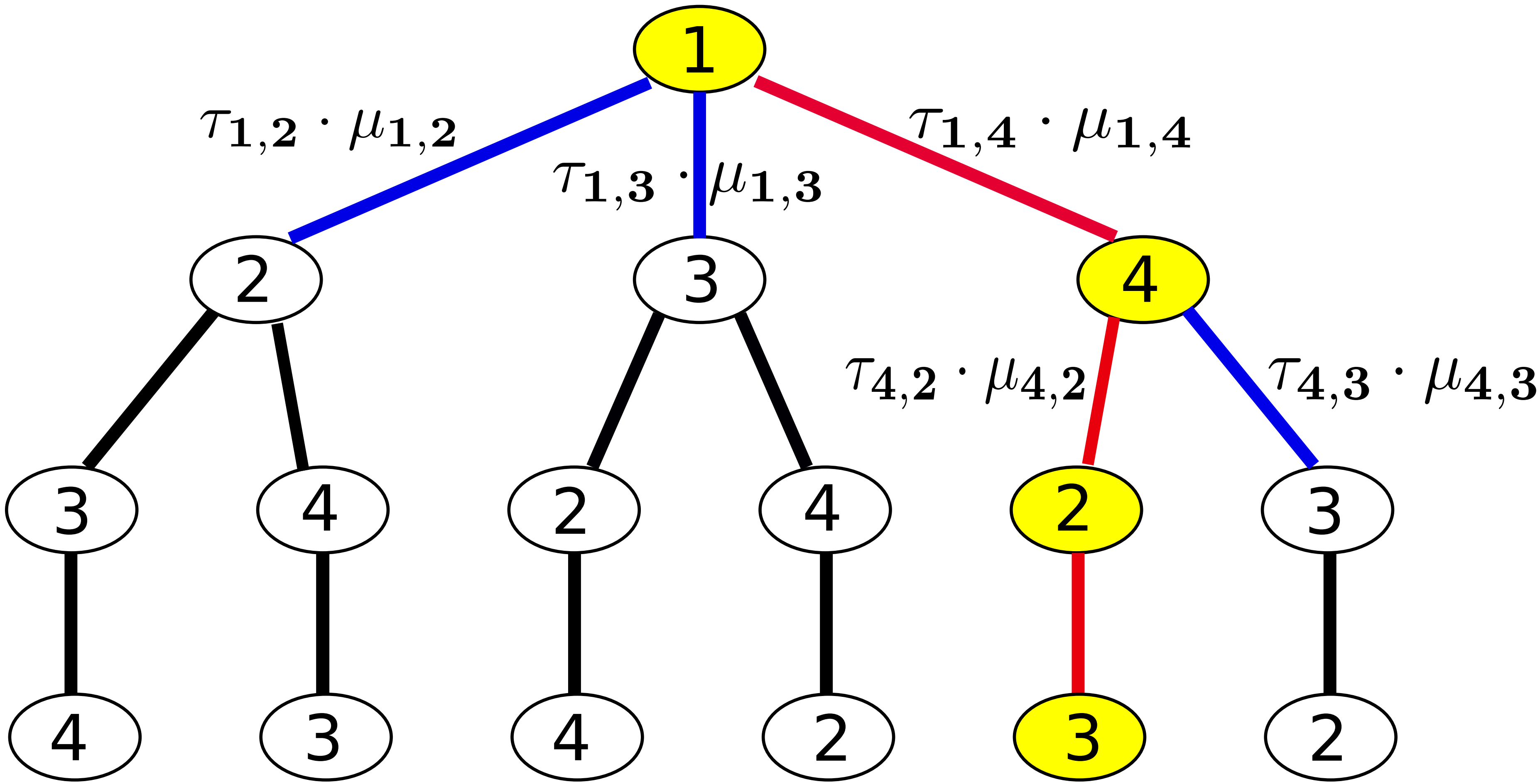
Beam-ACO

Short description

Ant Colony Optimization (ACO)

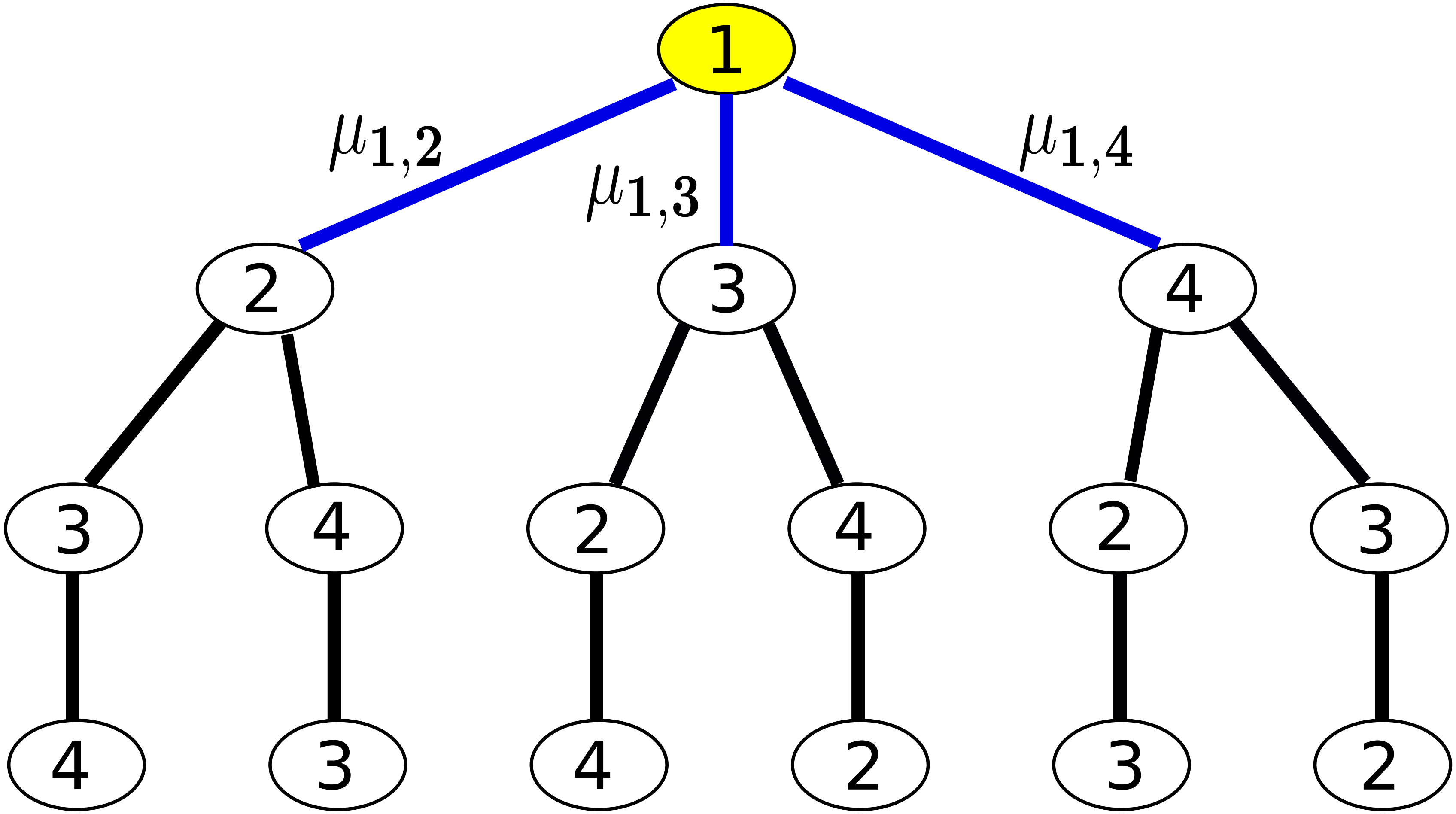


ACO is a tree search algorithm



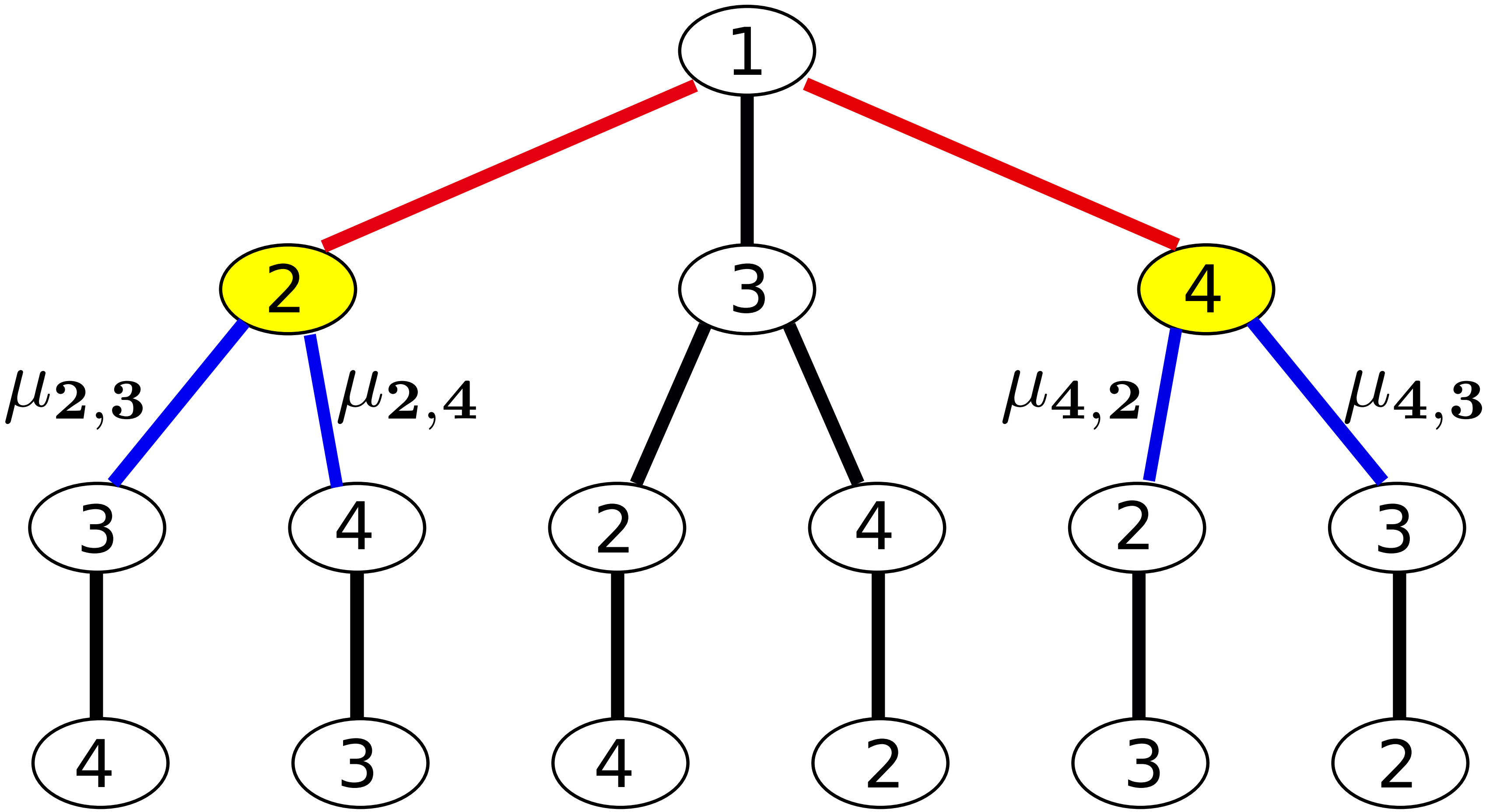
Beam search: 1st construction step

Parameters: $k_{ext} = 2$, $k_{bw} = 3$



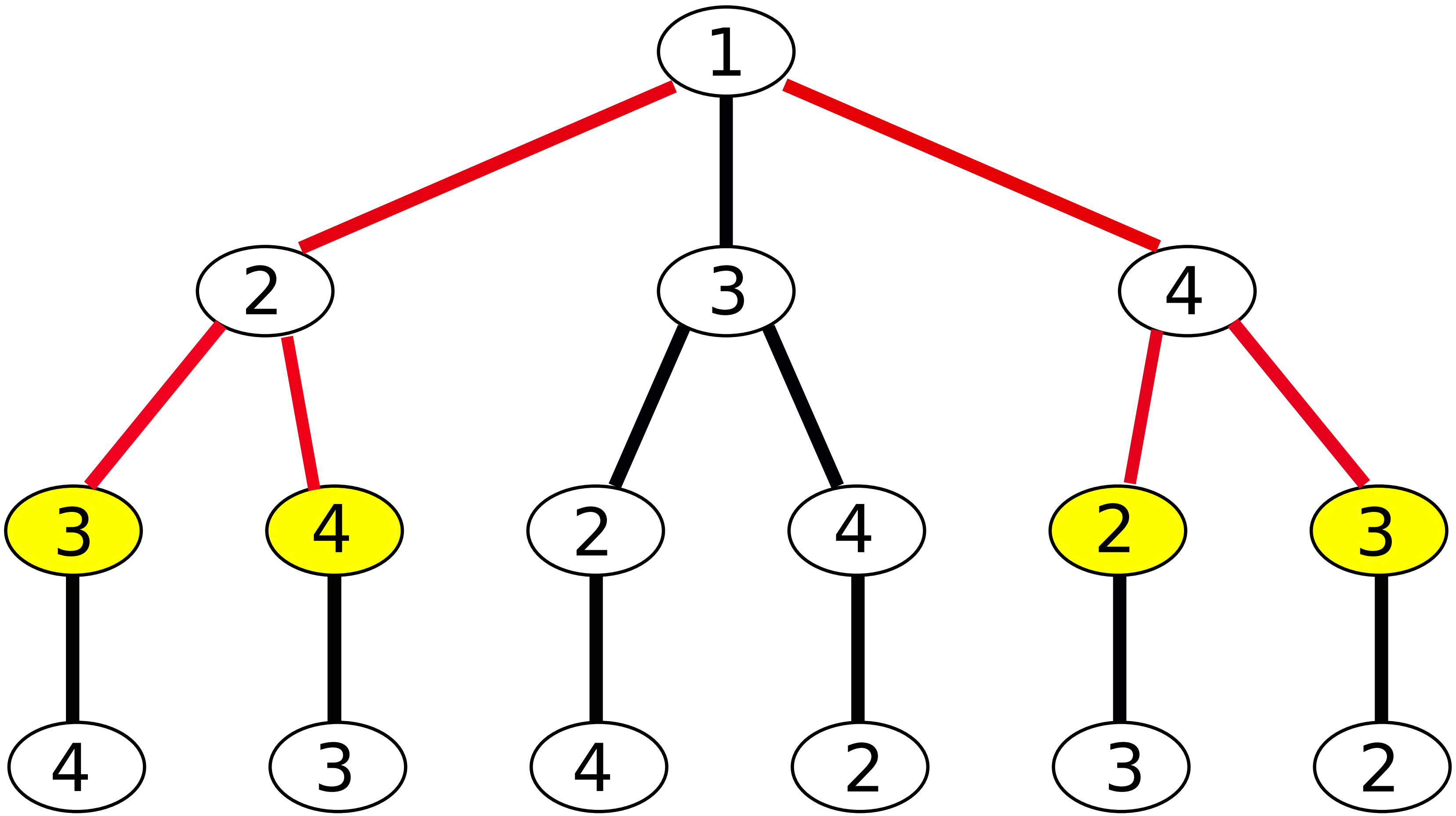
Beam search: 2nd construction step

$\boxed{\text{Parameters:}}$ $k_{ext} = 2$, $k_{bw} = 3$



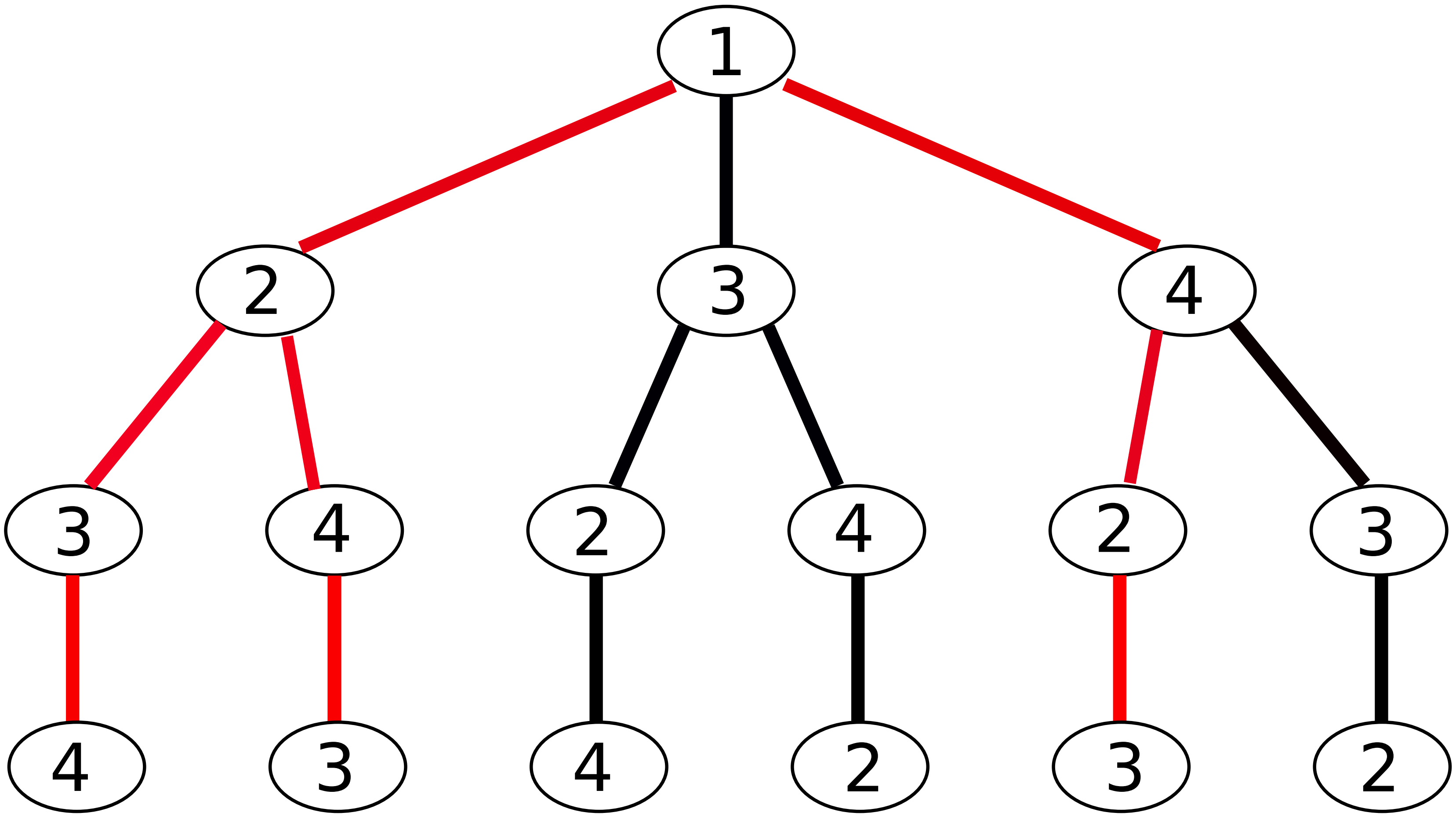
Beam search: after 2nd construction step

Use of: lower (upper) bound



Beam search: 3rd construction step

$\boxed{\text{Parameters:}}$ $k_{ext} = 2$, $k_{bw} = 3$



Hybrid algorithm: Beam-ACO

Idea:

- Instead of n_a independent solution constructions per iteration,
- perform a probabilistic beam search with beam width $k_{bw} = n_a$

Advantages:

- Strong heuristic guidance by a lower bound
- Embedded in the adaptive framework of ACO

Requirements for the lower bound:

- Fast to compute
- Differentiate well between nodes on the same level of the search tree

Hybrid algorithm: Beam-ACO

Applications Beam-ACO was applied to the following problems:

- **Open shop scheduling (OSS)**
  Blum, *Computers & Operations Research* (2005)
- **Supply chain management**
  Caldeira et al., *FUZZ-IEEE 2007*, *ISFA 2007*
- **Simple assembly line balancing (SALB)**
  Blum, *INFORMS Journal on Computing* (2008)
- **Travelling salesman problem with time windows (TSPTW)**
  López-Ibañez et al., *Computers & Operations Research* (2010)
- **Longest common subsequence (LCS) problems**
  Blum et al. *CEC 2010*, *EA 2013*, *Journal of Heuristics* (2016)
- **Weighted vehicle routing problem**
  Tang et al. *IEEE Transactions on Automation Science and Engineering* (2014)

Hybrid algorithm: Beam-ACO

Question: Why does it work so well?

Observation: Beam-ACO uses 2 types of complementary problem information

1. A greedy function
2. Lower (respectively, upper) information

These two types of information are especially well exploited in Beam-ACO!

Construct, Merge, Solve & Adapt (CMSA)

Short description

Why combining metaheuristics with ILP Solvers?

General advantage of metaheuristics:

- Very good in exploiting information on the problem (greedy heuristics)
- Generally very good in obtaining high-quality solutions for medium and even large size problem instances

However:

- Metaheuristics may also reach their limits with growing problem instance size
- Metaheuristics fail when the information on the problem is misleading

Goal: Taking profit from valuable optimization expertise that went into the development of ILP solvers even in the context of large problem instances

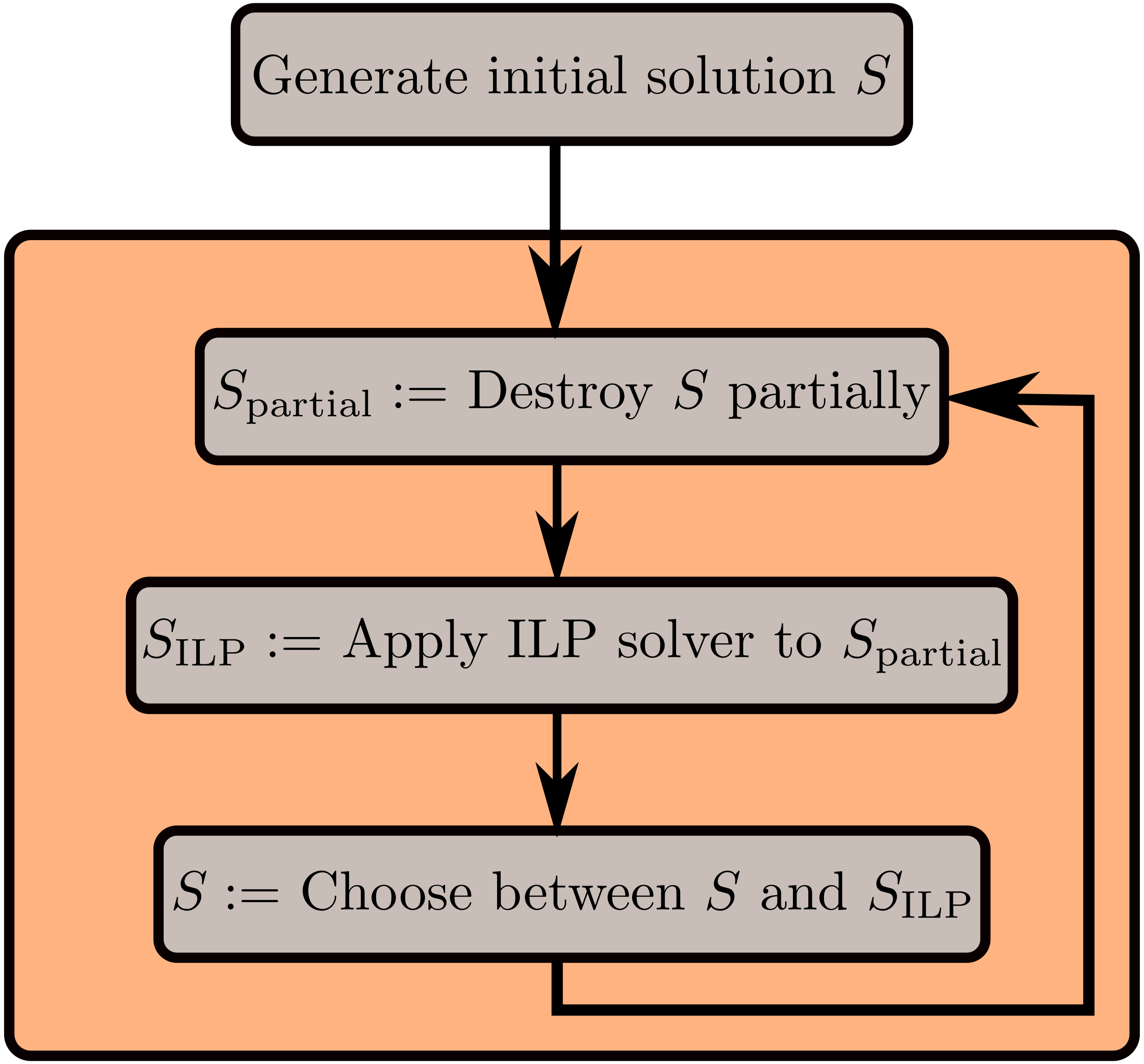
Standard: Large Neighborhood Search

- Small neighborhoods:
  1. **Advantage:** It is fast to find an improving neighbor (if any)
  2. **Disadvantage:** The average quality of the local minima is low
- Large neighborhoods:
  1. **Advantage:** The average quality of the local minima is high
  2. **Disadvantage:** Finding an improving neighbor might itself be NP-hard due to the size of the neigbhorhood

Ways of examining large neighborhoods:

- Heuristically
- Exact techniques: for example an ILP solver

ILP-based large neighborhood search: ILP-LNS



Hypothesis and resulting research question

In our experience: LNS works especially well when

1. The **number of solution components** (variables) is is not high
2. The **number of components in a solution** is not too small

Question:

What kind of general algorithm can we apply when the above conditions are not fullfilled?

Construct, Merge, Solve & Adapt: Principal Idea

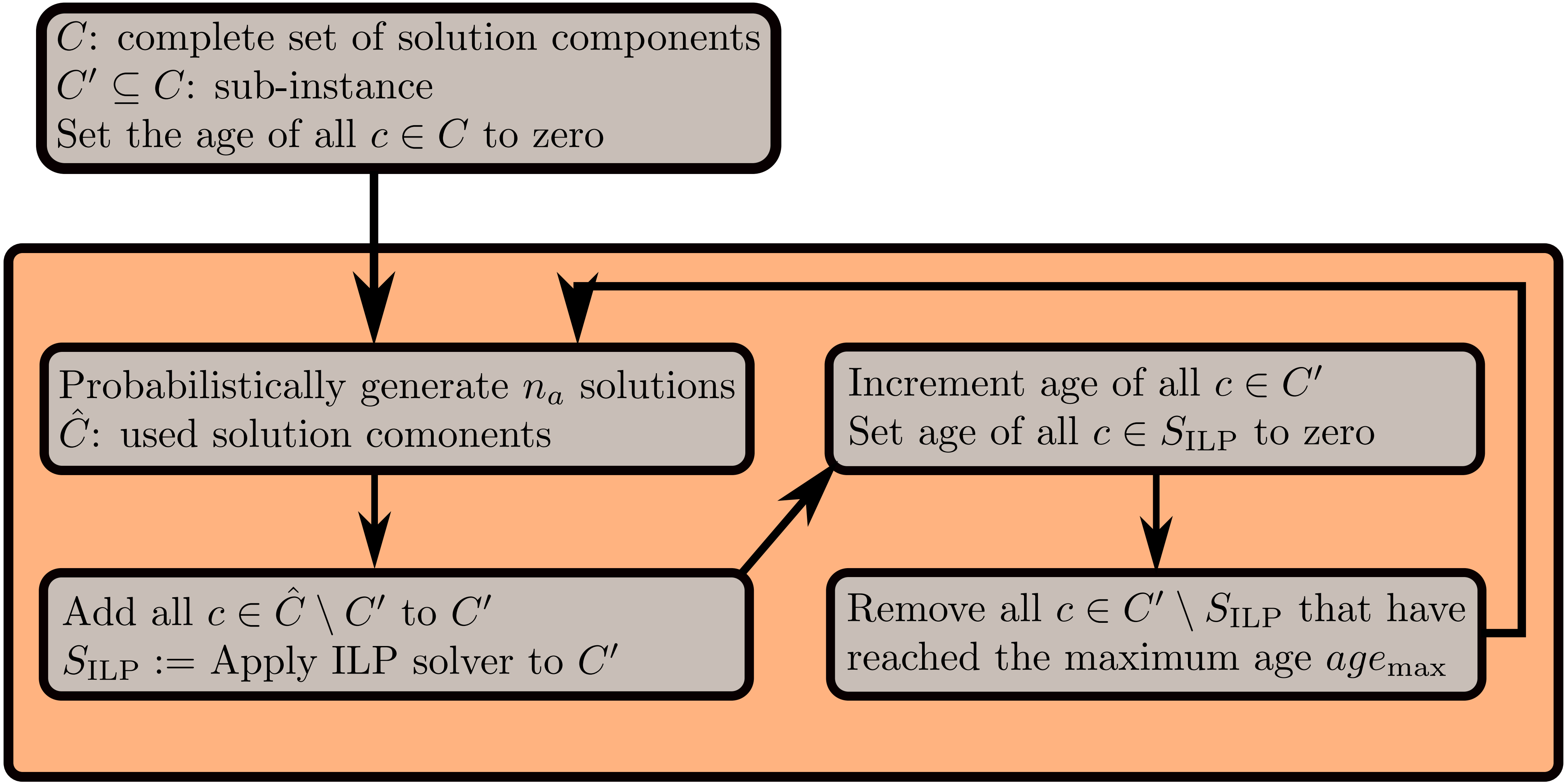
Observation: In the presence of a large number of solutions components, many of them only lead to bad solutions

Idea: Exclude the presumably bad solution components from the ILP

Steps of the proposed method:

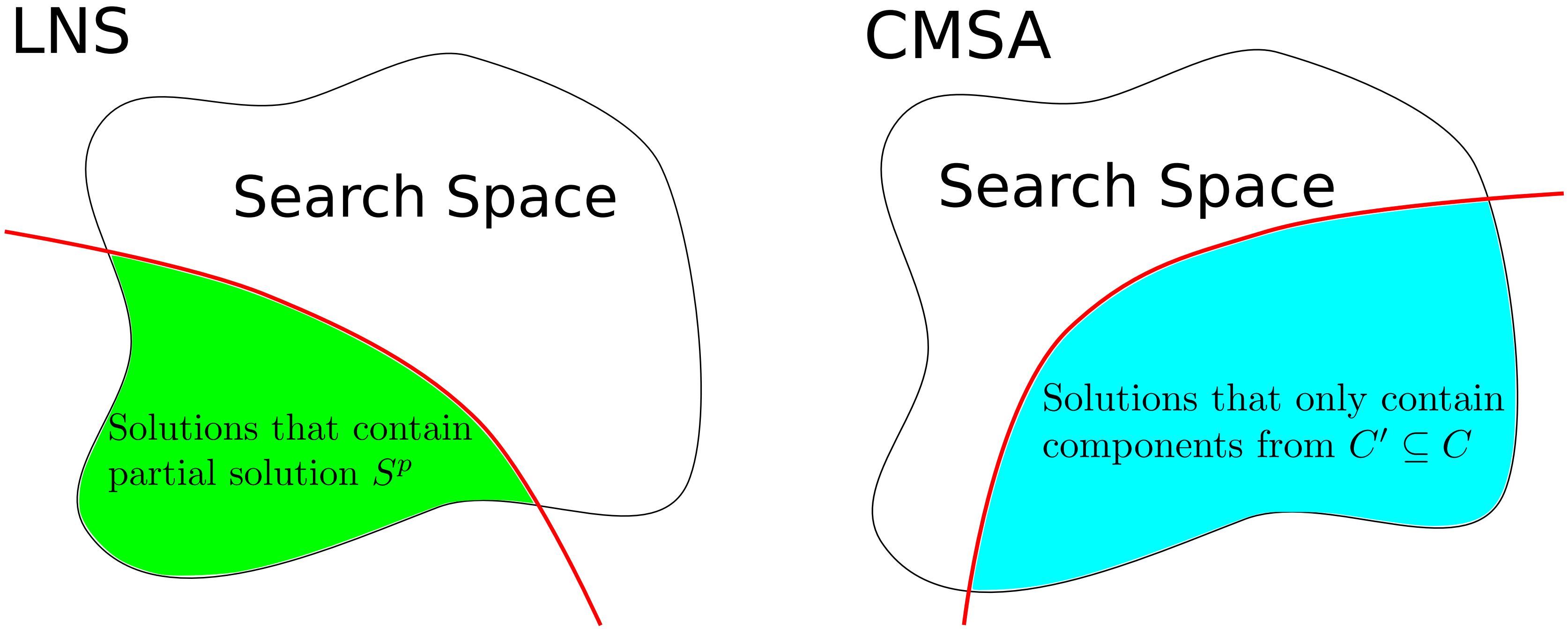
- Iteratively generate presumably good solutions in a probabilistic way
- Assemble a sub-instance from the used solution components
- Solve the sub-instance by means of an ILP solver
- Delete useless solution components from the sub-instance

Construct, Merge, Solve & Adapt: Flow Diagram



Differences between LNS and CMSA: summarized

How is the original problem instance reduced?



How is the sub-instance of the next iteration generated?

- LNS: Partial destruction of the incumbent solution
- CMSA: Generating new solutions and removing **old** solution components

Longest common subsequence (LCS) problem (1)

Notation: What is a subsequence of a string?

A string t is called a subsequence of a string x,
iff t can be produced from x by deleting characters

Example: Is AAT a subsequence of ACAGTTA?

ACAGTTA

Longest common subsequence (LCS) problem (2)

Problem definition (restricted to two input sequence)

Given: A problem instance (x, y, Σ), where

- x and y are input sequences over the alphabet Σ

Optimization goal:

Find a longest string t^* that is a subsequence of strings x and y $\rightarrow$ a longest common subsequence

Repetition-free longest common subsequence problem

- Restriction: No letter **may appear more than once** in a valid solution
- Proposed in: 2010 in *Discrete Applied Mathematics*
- Hardness: APX-hard (shown in above paper)
- Motivation: Genome rearrangement where duplicate genes are basically not considered
- Existing algorithms:
  1. Three simple heuristics, *Discrete Applied Mathematics*, 2010
  2. An Evolutionary Algorithm, *Operations Research Letters*, 2013

A simple constructive RFLCS heuristic: Best-Next (1)

Principle: Builds a solution sequentially from left to right

1: **input:** a problem instance (x, y, Σ)
2: **initialization:** $t := \epsilon$ (where ϵ is the empty string)
3: **while** $|\Sigma_t^{\mathrm{nd}}| > 0$ **do**
4: $\quad a :=$ $\mathsf{ChooseFrom}(\Sigma_t^{\mathrm{nd}})$
5: $\quad t := ta$
6: **end while**
7: **output:** a repetition-free common subsequence t

Question: How is Σ_t^{nd} defined?

A simple constructive LCS heuristic: Best-Next (2)

Example: Given is

- **Problem instance** $(x, y, \Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{T}, \mathrm{G}\})$ where
  - $x =$ ATCTAGCTG
  - $y =$ TACCATGTG
- **Partial solution** $t =$ AC

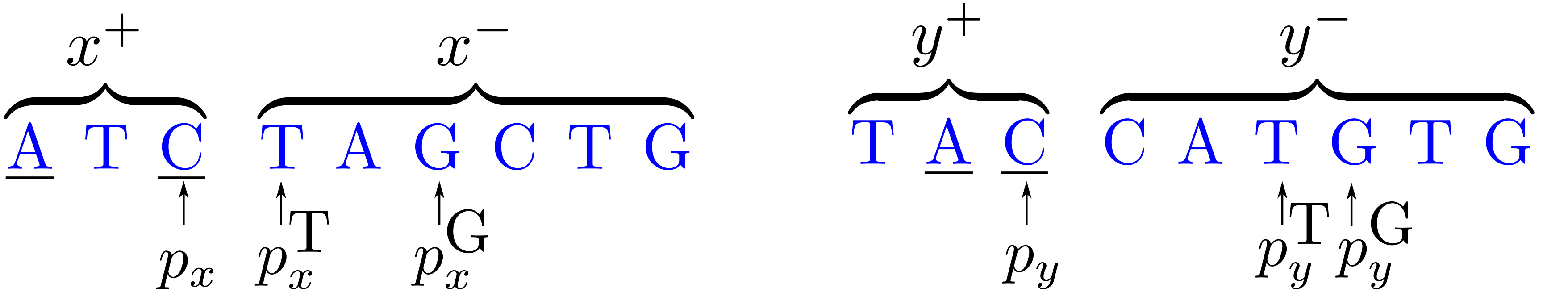
$$\overbrace{\underline{\mathrm{A}}\ \mathrm{T}\ \underset{\substack{\uparrow\\ p_x}}{\underline{\mathrm{C}}}}^{x^+}\quad \overbrace{\underset{\substack{\uparrow\\ p_x^{\mathrm{T}}}}{\mathrm{T}}\ \mathrm{A}\ \underset{\substack{\uparrow\\ p_x^{\mathrm{G}}}}{\mathrm{G}}\ \mathrm{C}\ \mathrm{T}\ \mathrm{G}}^{x^-}\qquad\qquad \overbrace{\mathrm{T}\ \underline{\mathrm{A}}\ \underset{\substack{\uparrow\\ p_y}}{\underline{\mathrm{C}}}}^{y^+}\quad \overbrace{\mathrm{C}\ \mathrm{A}\ \underset{\substack{\uparrow\\ p_y^{\mathrm{T}}}}{\mathrm{T}}\ \underset{\substack{\uparrow\\ p_y^{\mathrm{G}}}}{\mathrm{G}}\ \mathrm{T}\ \mathrm{G}}^{y^-}$$

Result: $\Sigma_t^{\mathrm{nd}} = \{\mathrm{T}\}$

Greedy function

Greedy function:

$$\eta(ta) := \left(\frac{p_x^a - p_x}{|x^-|} + \frac{p_y^a - p_y}{|y^-|} \right)^{-1}, \quad \forall a \in \Sigma_t^{\mathrm{nd}}$$



Pheromone model

- $\tau_{x,i}$: desirability to add the letter at position i of string x to the solution
- $\tau_{y,i}$: desirability to add the letter at position i of string y to the solution

Transition probabilities in Beam-ACO: given partial solution t,

$$\mathbf{p}(ta) = \frac{\left(\min\{\tau_{x,p_x^a}, \tau_{y,p_y^a}\} \cdot greedyinfo\right)}{\sum_{b \in \Sigma_t^{nd}} \left(\min\{\tau_{x,p_x^b}, \tau_{y,p_y^b}\} \cdot greedyinfo\right)} \quad , \forall\, a \in \Sigma_t^{nd}$$

Upper bound function

Given a partial solution t:

- Each input string x is partitioned into
  1. $x^+ :=$ first part of x until p_x
  2. $x^- :=$ remaining part of x (after p_x)
- $\delta(a, x)$ evaluates to 1 if letter a appears at least once in x^-, to 0 otherwise.

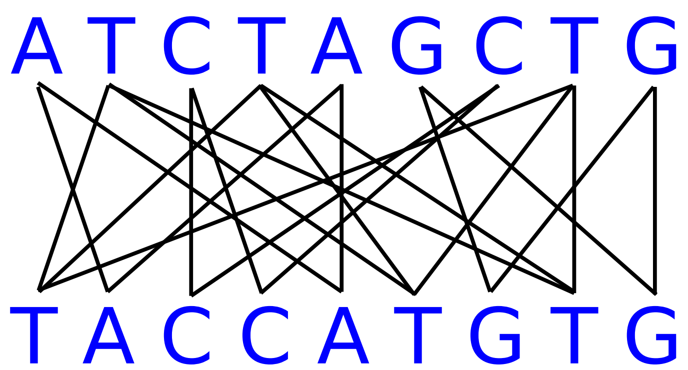
$$\mathrm{UB}(t) := |t| + \sum_{a \in \Sigma_t} \min\{\delta(a, x), \delta(a, y)\}$$

ILP Model (1)

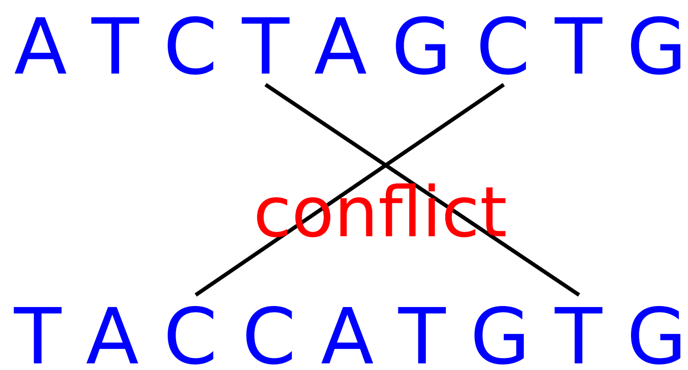
Set of binary variables:

For each position i of x and j of y such that $x[i] = y[j]$ the model has a variable $z_{i,j}$

Example set of variables



Example of a conflict



ILP Model (2)

$$\textbf{max} \sum_{z_{i,j} \in Z} z_{i,j} \tag{1}$$

subject to:

$$\sum_{z_{i,j} \in Z_a} z_{i,j} \leq 1 \quad \text{for } a \in \Sigma \tag{2}$$

$$z_{i,j} + z_{k,l} \leq 1 \quad \text{for all } z_{i,j} \text{ and } z_{k,l} \text{ being in conflict} \tag{3}$$

$$z_{i,j} \in \{0,1\} \quad \text{for } z_{i,j} \in Z \tag{4}$$

Hereby:

- $z_{i,j} \in Z_a$ iff $x[i] = y[j] = a$
- $z_{i,j}$ and $z_{k,l}$ are in conflict iff $i < k$ and $j > l$ OR $i > k$ and $j < l$

Experimental evaluation: benchmark instances

Set1: 30 instances for each combination of

- Input sequence length: $n \in \{32, 64, 128, 256, 512, 1024, 2028, 4048\}$
- Alphabet size: $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$

Set2: 30 instances for each combination of

- Alphabet size: $|\Sigma| \in \{4, 8, 16, 32, 64, 128, 256, 512\}$
- Maximal number of repetitions of each letter: $rep \in \{3, 4, 5, 6, 7, 8\}$

Tuning: CMSA's and BEAM-ACO's parameters are tuned by irace for each alphabet size

Experimental results: performance of CPLEX

Set1:

- Input sequence length: $n \in \{\mathbf{32}, \mathbf{64}, \mathbf{128}, \mathbf{256}, \mathbf{512}, 1024, 2028, 4048\}$
- Alphabet size: $|\Sigma| \in \{\mathbf{n/8}, \mathbf{n/4}, \mathbf{3n/8}, \mathbf{n/2}, 5n/8, 3n/4, 7n/8\}$

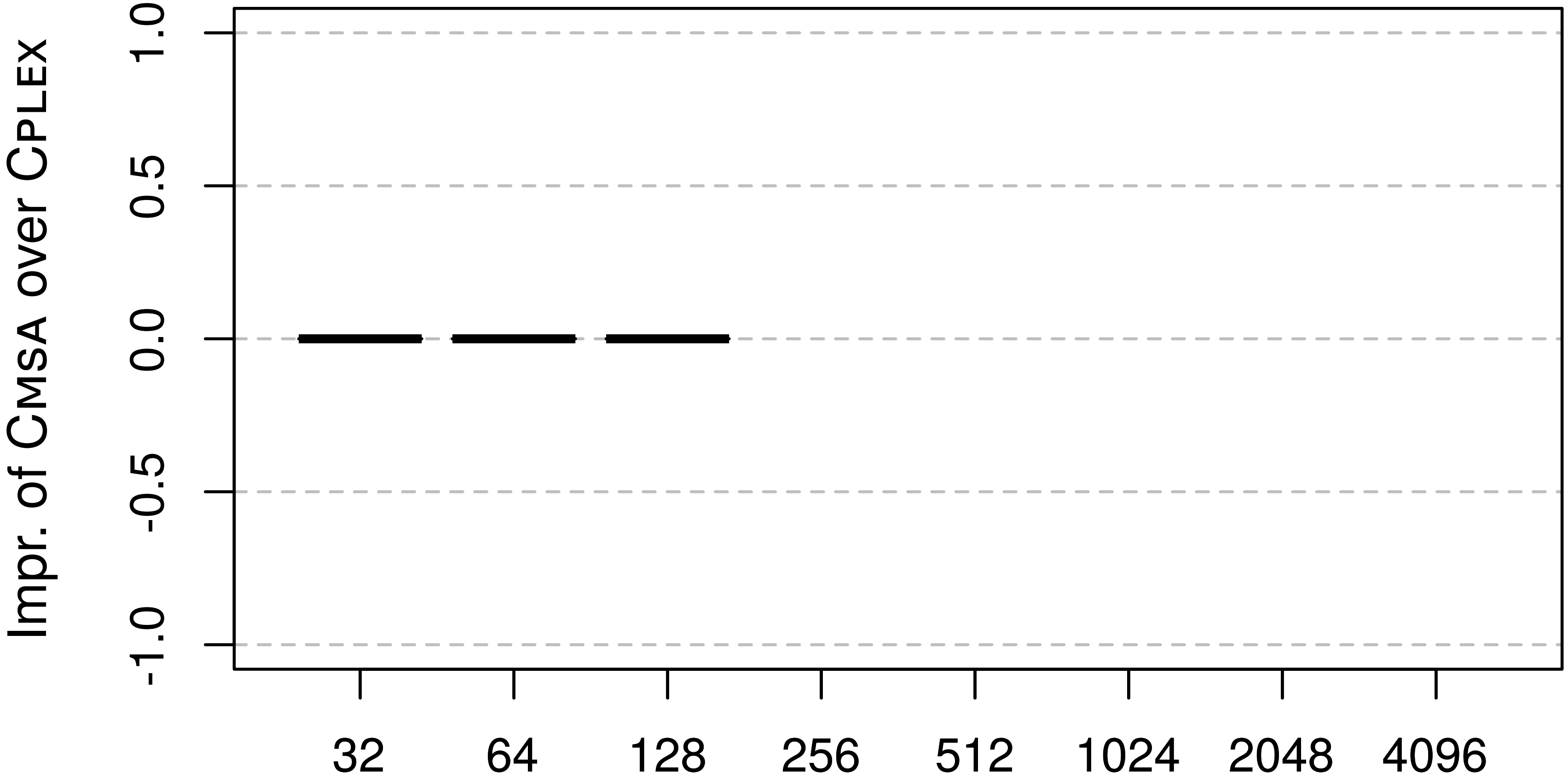
Set2:

- Alphabet size: $|\Sigma| \in \{\mathbf{4}, \mathbf{8}, \mathbf{16}, \mathbf{32}, \mathbf{64}, 128, 256, 512\}$
- Maximal number of repetitions of each letter: $rep \in \{3, 4, 5, \mathbf{6}, \mathbf{7}, \mathbf{8}\}$

Result: CPLEX is able to solve nearly all exisiting problem instances from the literature to optimality

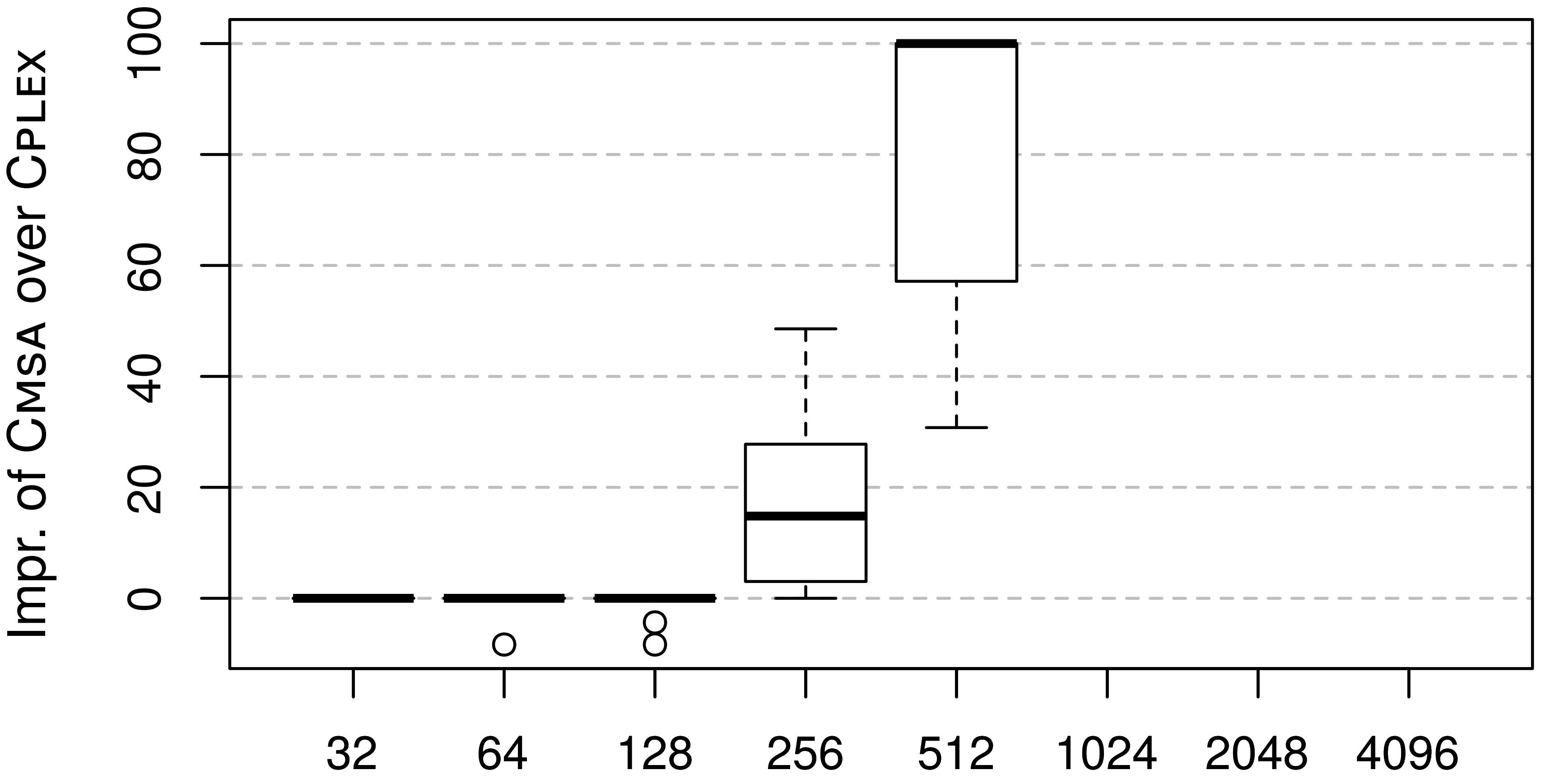
Experimental results: Set1

Improvement of CMSA over CPLEX: alphabet size $n/8$



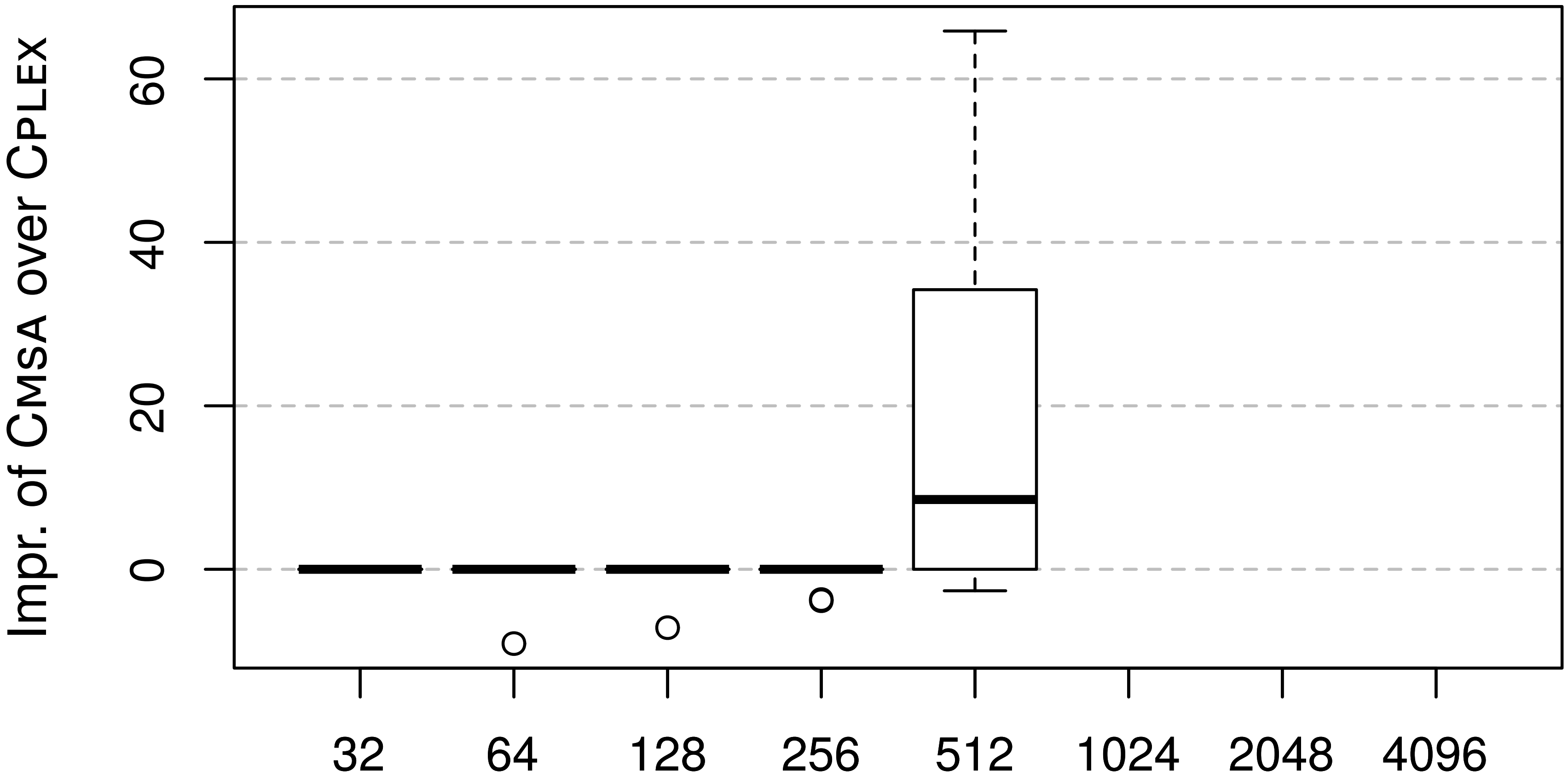
Experimental results: Set1

Improvement of CMSA over CPLEX: alphabet size $n/2$



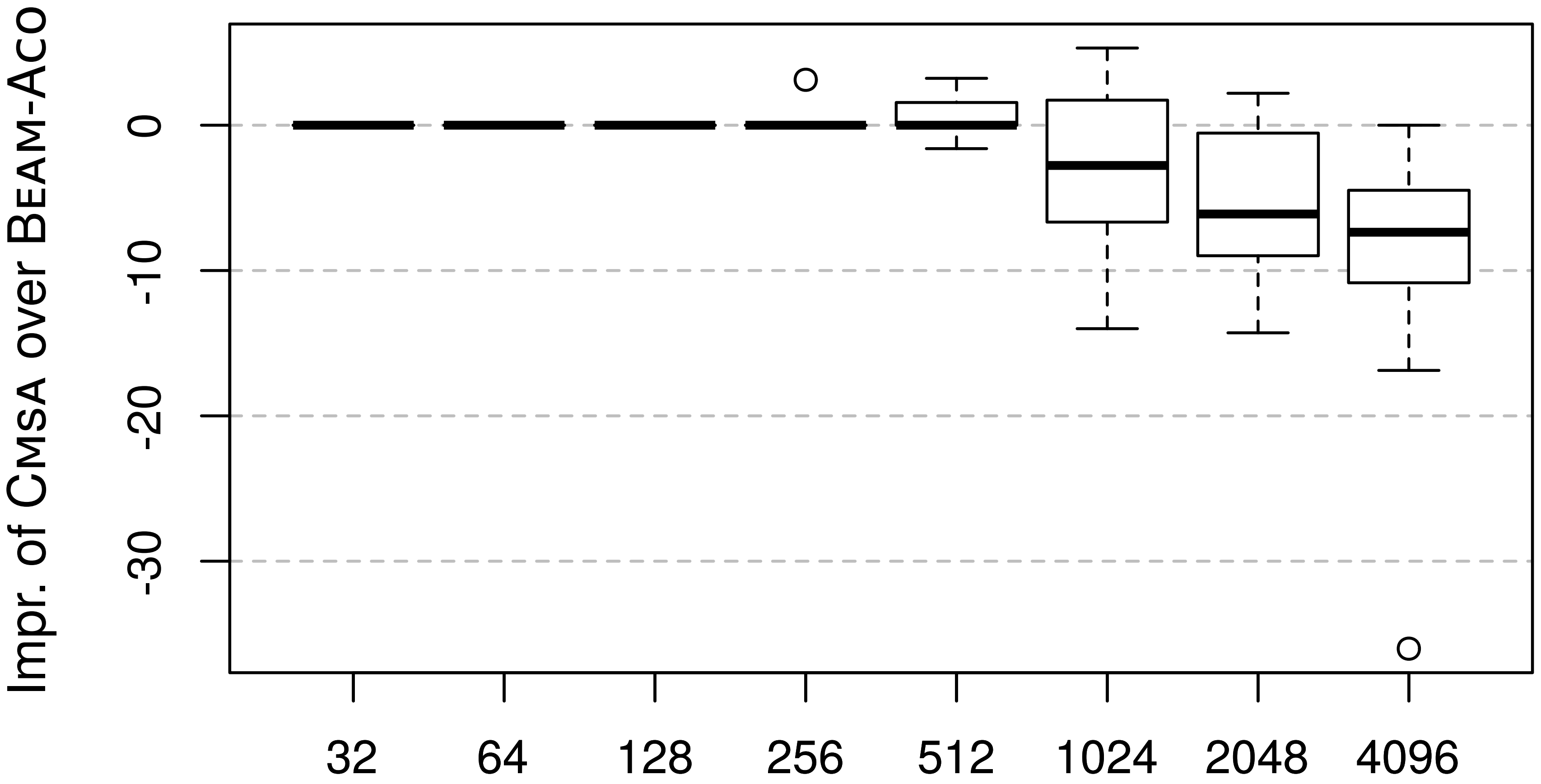
Experimental results: Set1

Improvement of CMSA over CPLEX: alphabet size $7n/8$



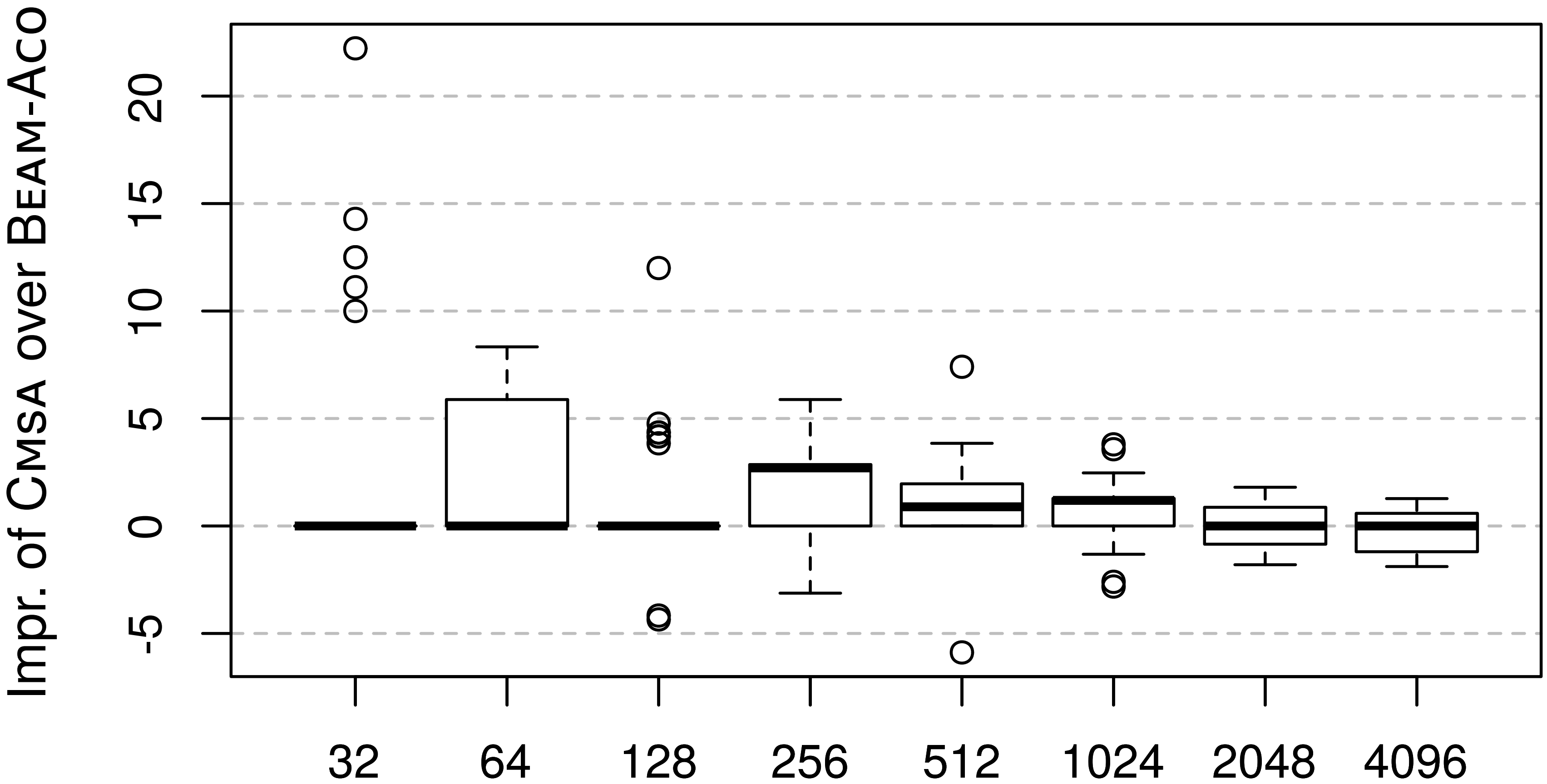
Experimental results: Set1

Improvement of CMSA over Beam-ACO: alphabet size $n/8$



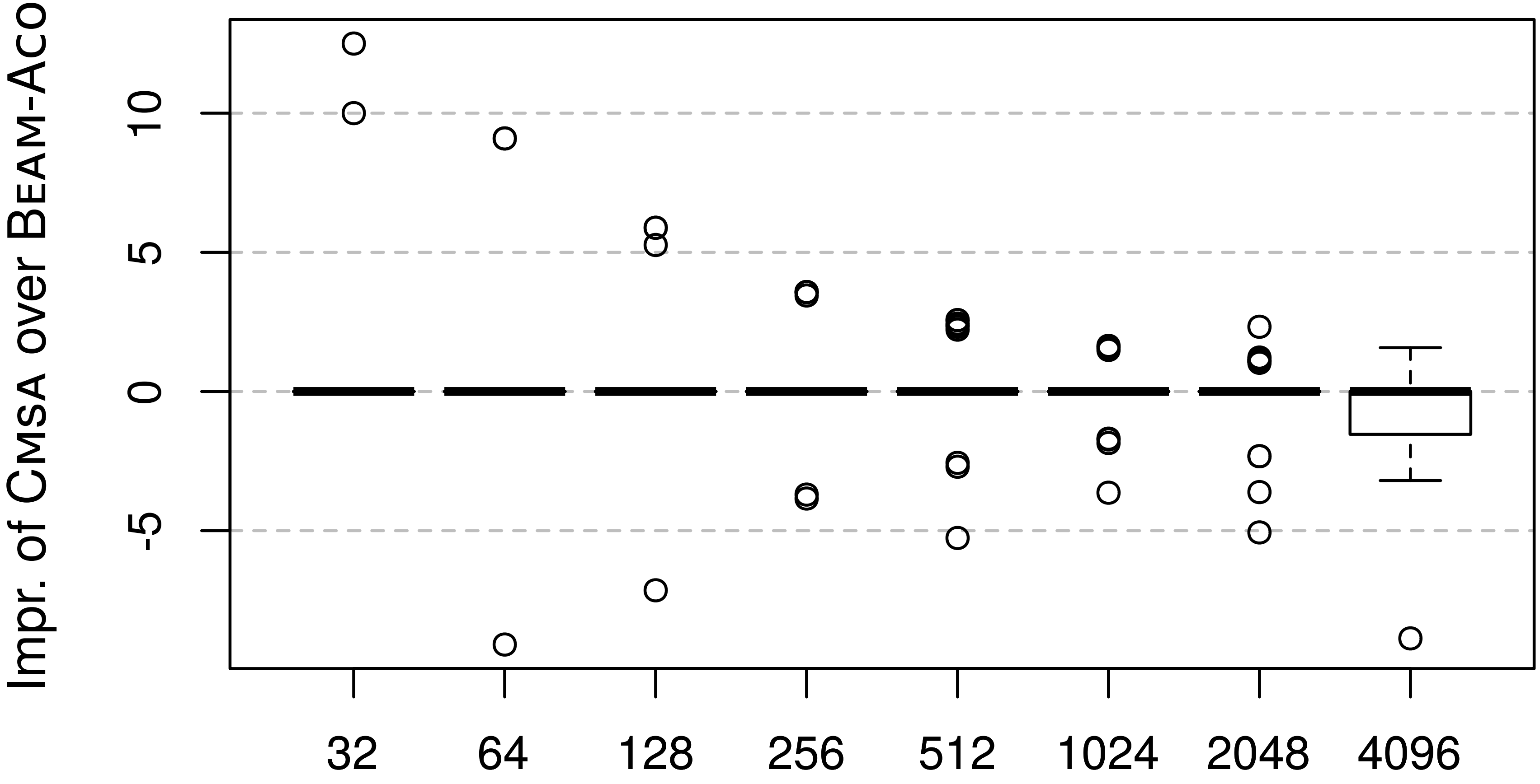
Experimental results: Set1

Improvement of CMSA over Beam-ACO: alphabet size $n/2$



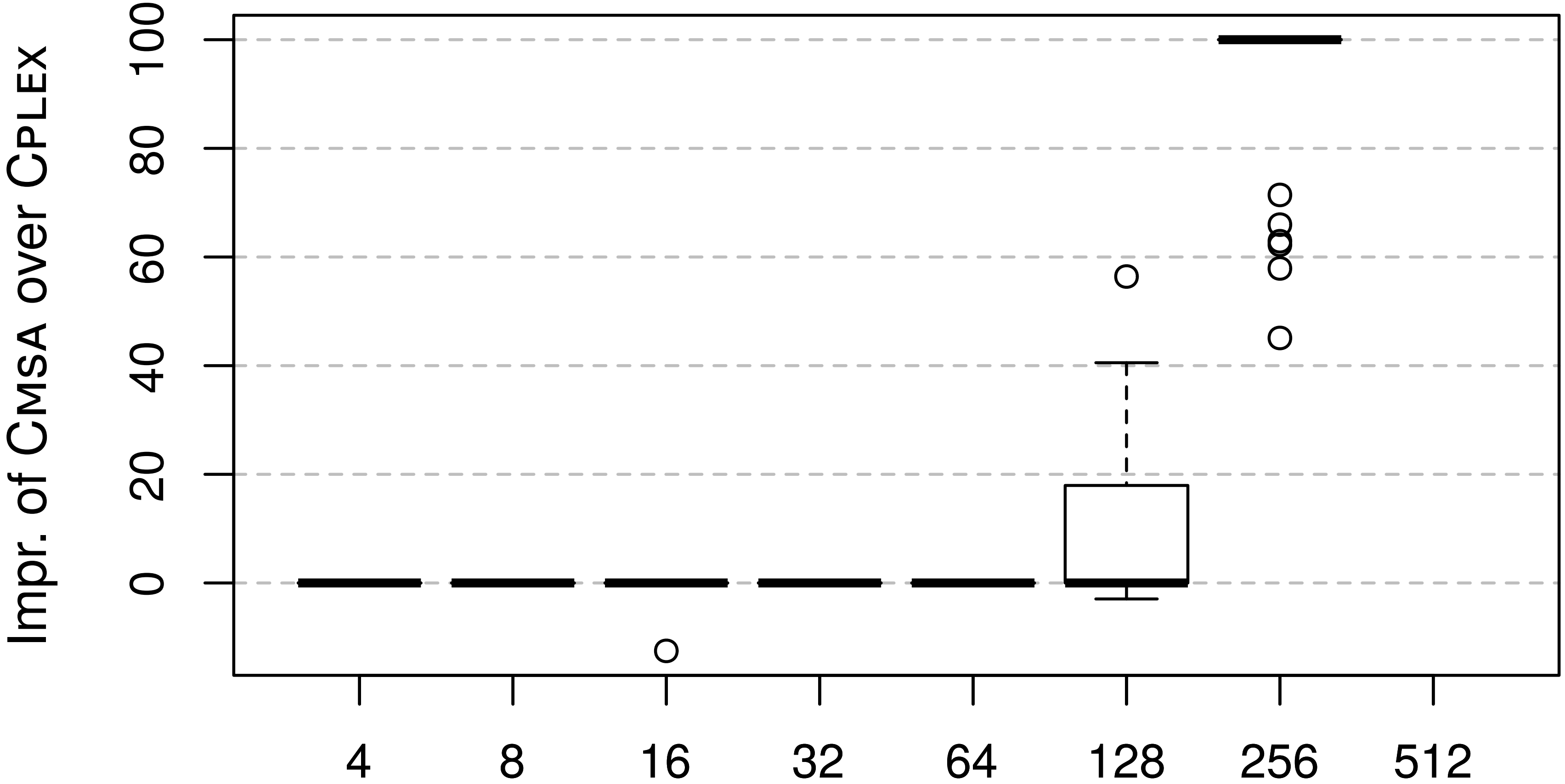
Experimental results: Set1

Improvement of CMSA over Beam-ACO: alphabet size $7n/8$



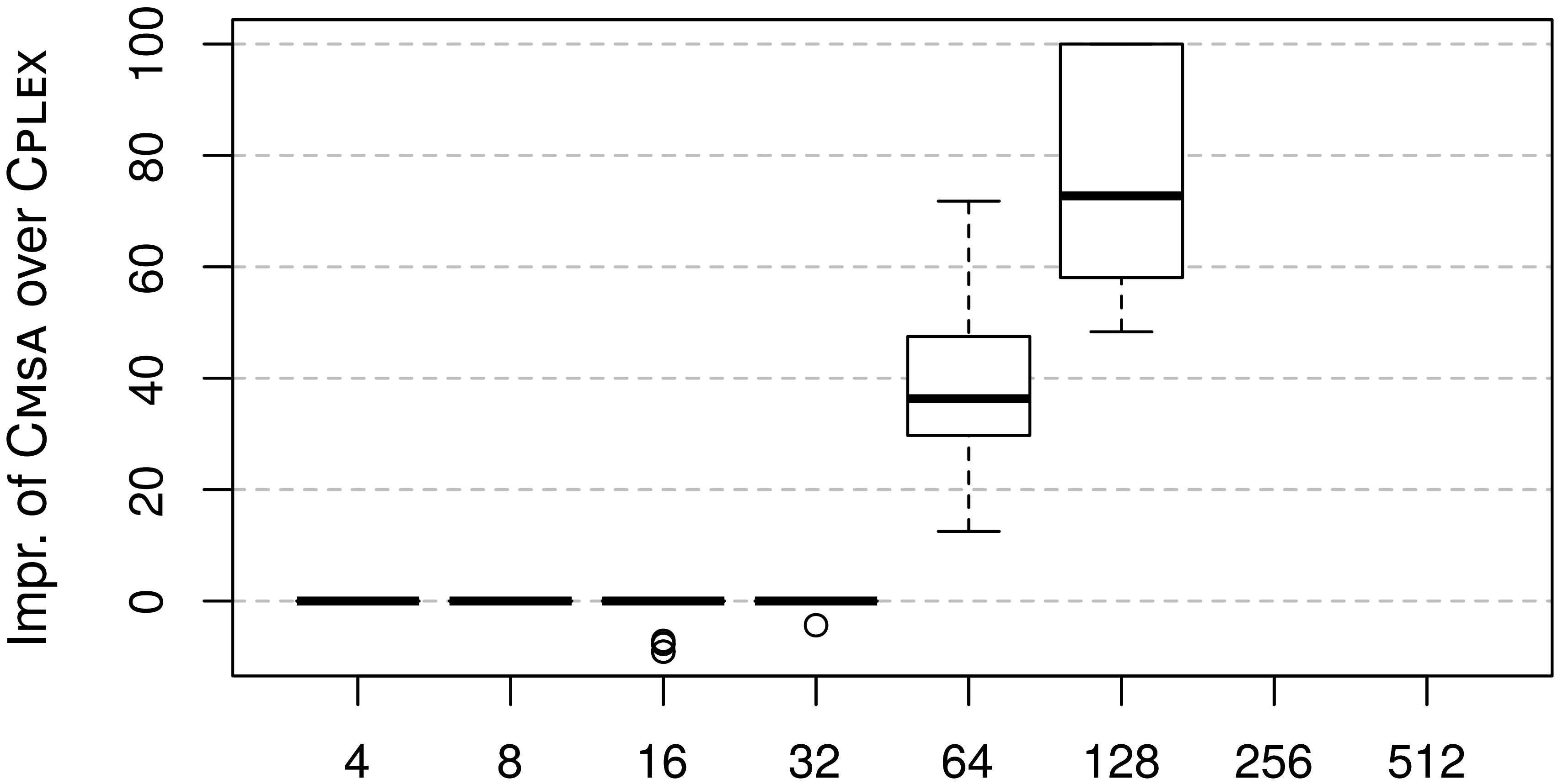
Experimental results: Set2

Improvement of CMSA over CPLEX: 3 reps



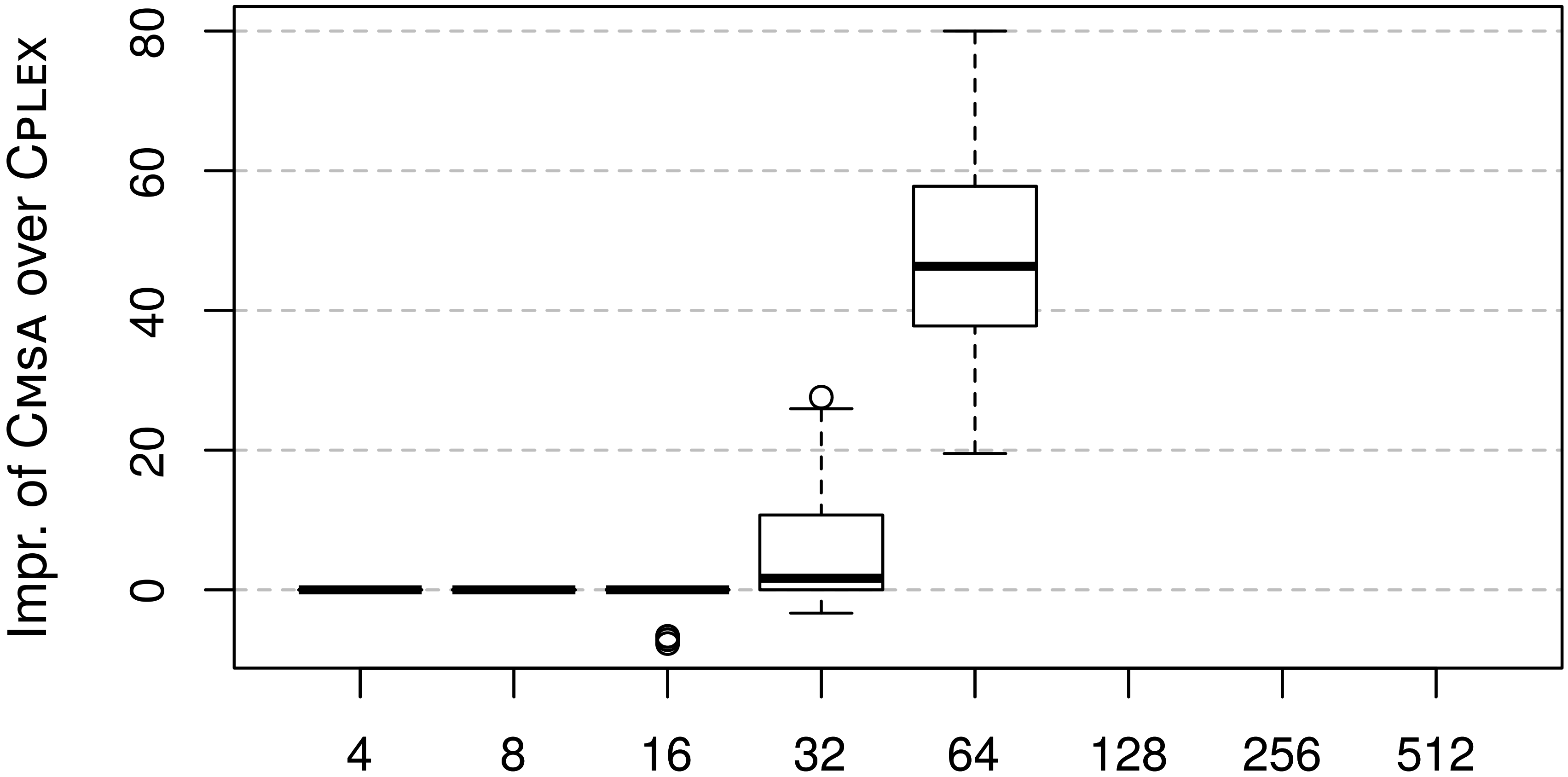
Experimental results: Set2

Improvement of CMSA over CPLEX: 6 reps



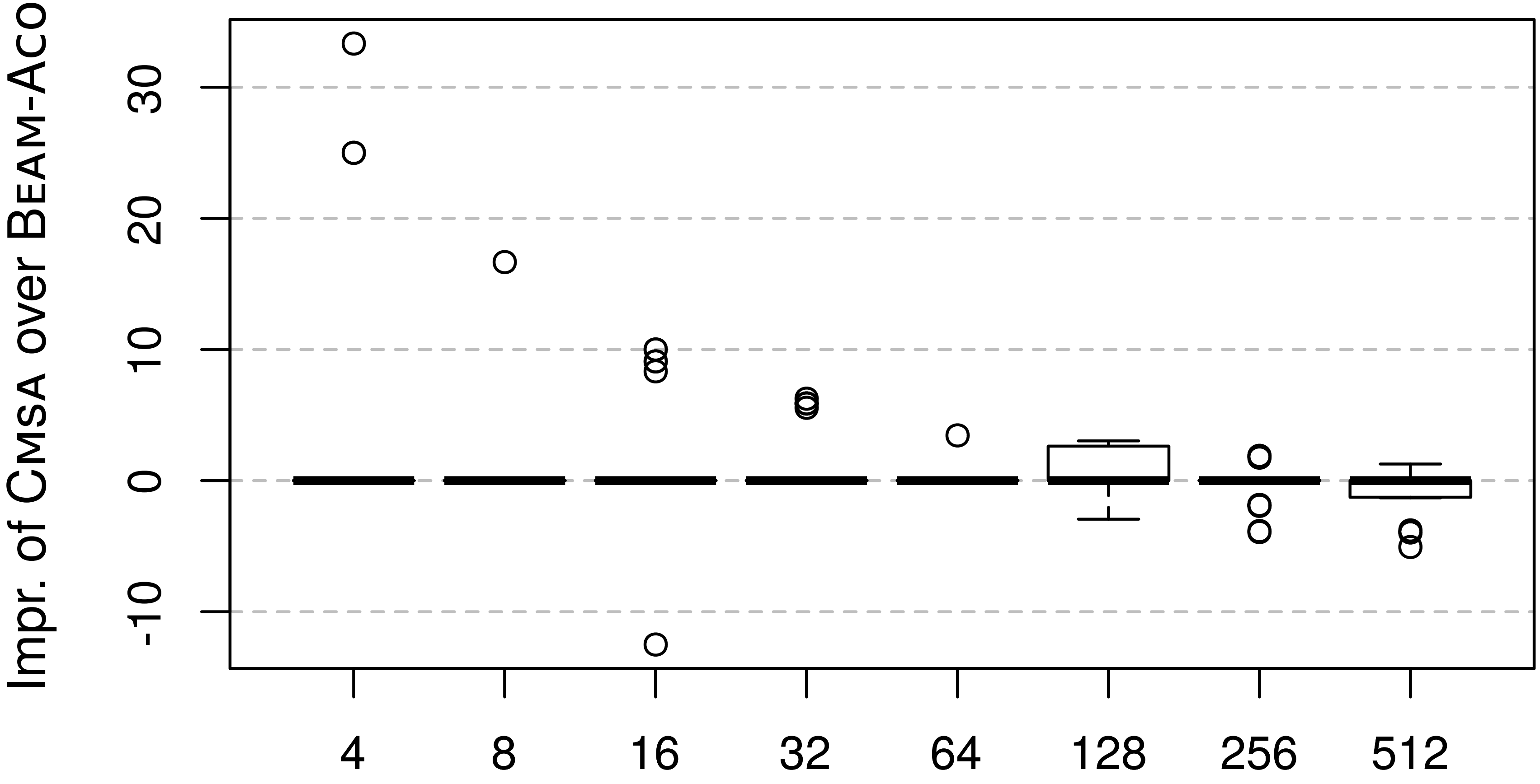
Experimental results: Set2

Improvement of CMSA over CPLEX: 8 reps



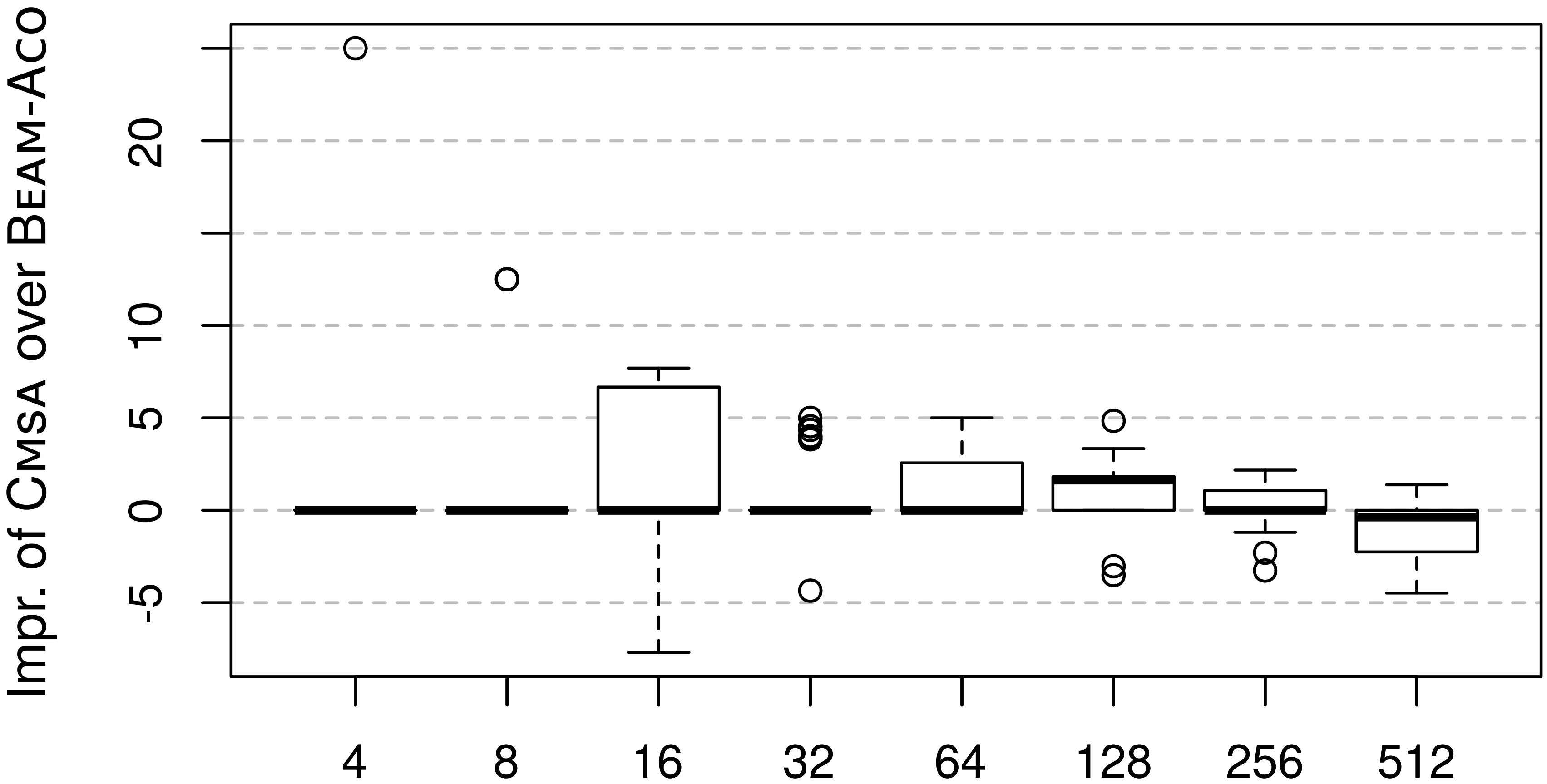
Experimental results: Set2

Improvement of CMSA over Beam-ACO: 3 reps



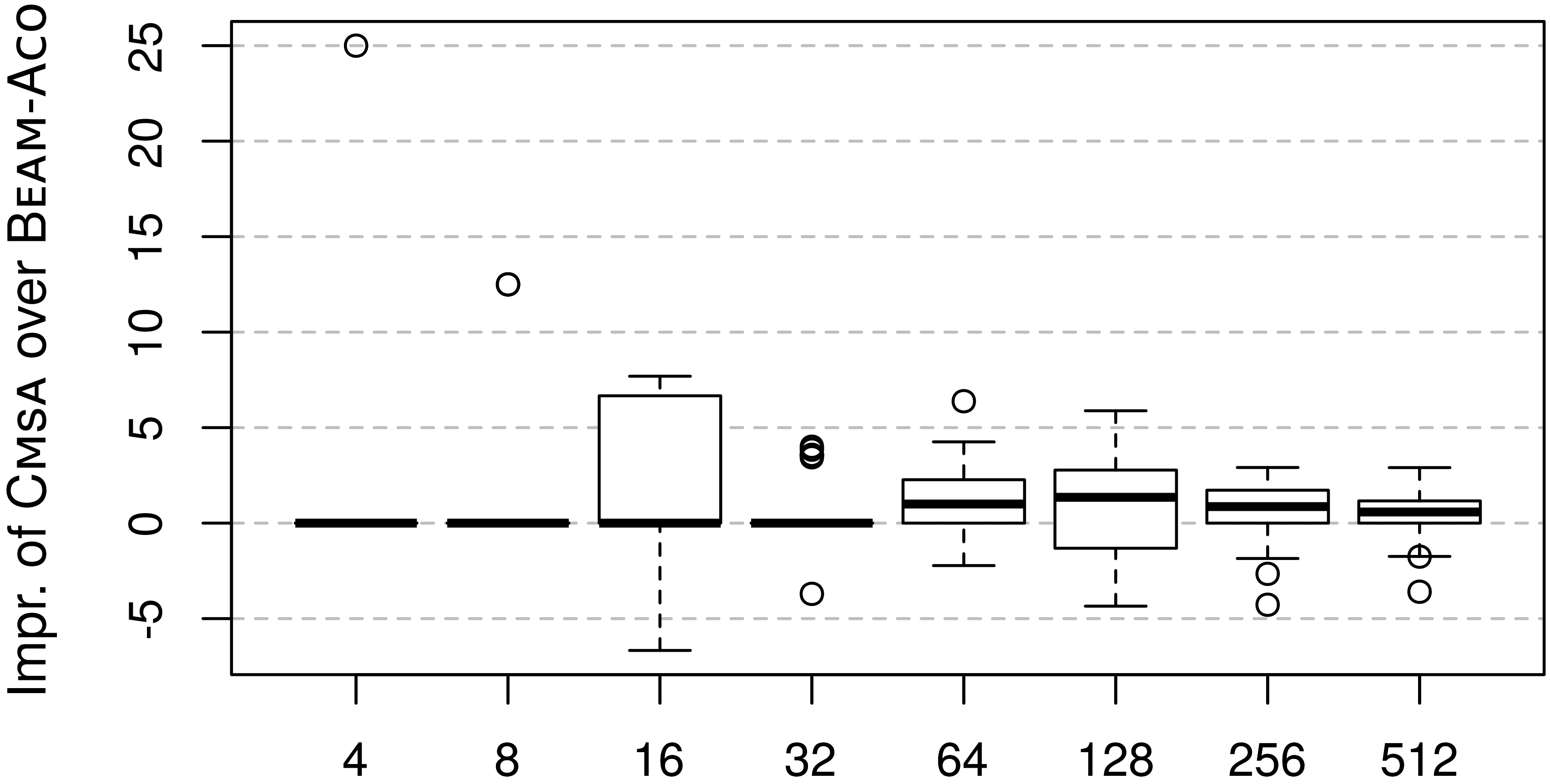
Experimental results: Set2

Improvement of CMSA over Beam-ACO: 6 reps

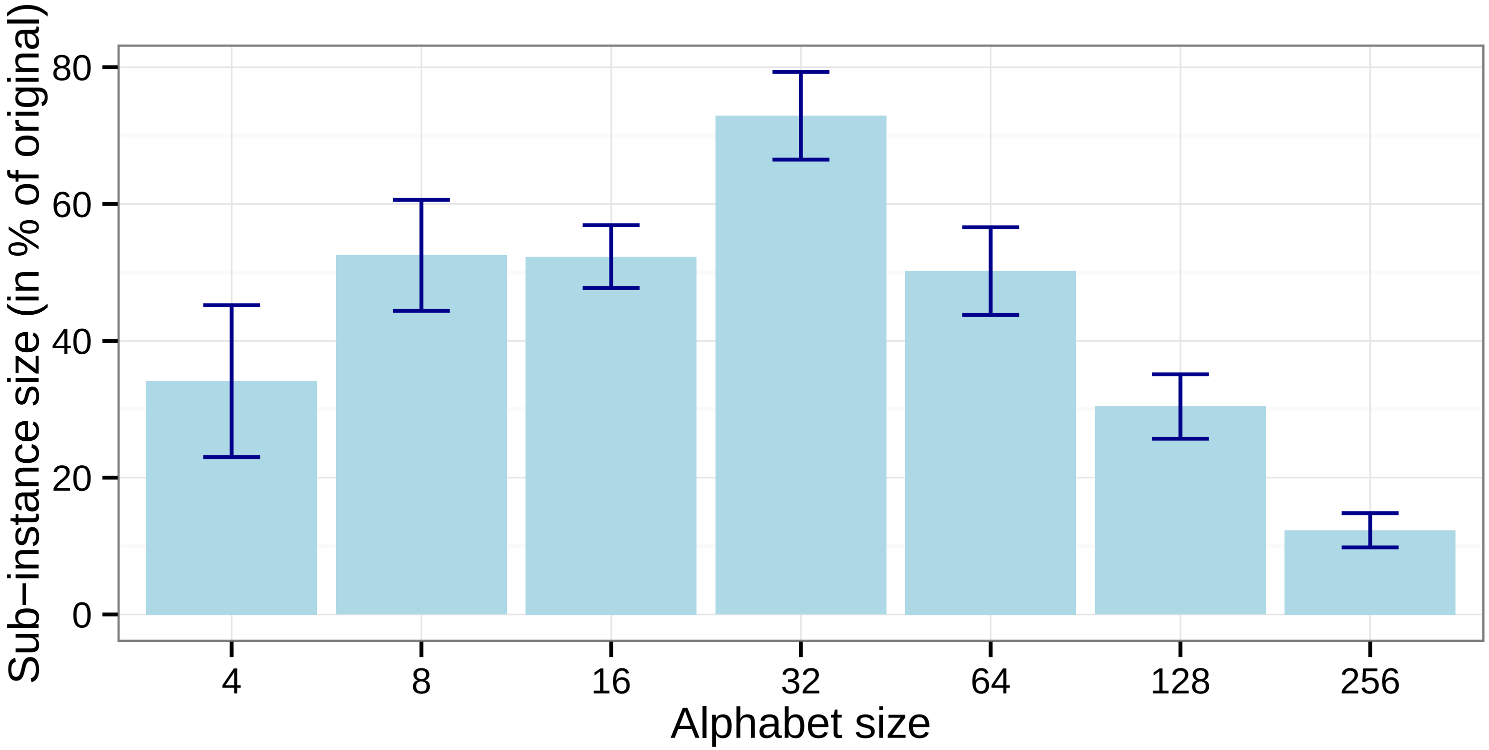


Experimental results: Set2

Improvement of CMSA over Beam-ACO: 8 reps



Experimental results: size of sub-instances



Synergy effects: Set1

$n\|\Sigma\|$	$n/8$	$n/4$	$3n/8$	$n/2$	$5n/8$	$3n/4$	$7n/8$
32	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64	0.0	0.0	0.0	0.0	0.03	0.0	0.04
128	0.0	0.07	0.0	0.07	0.0	0.0	0.03
256	0.0	-0.07	0.04	0.13	0.03	0.1	0.07
512	-0.14	0.2	0.17	0.27	0.23	0.23	0.14
1024	-0.4	0.47	0.23	0.53	0.03	0.44	0.13
2048	-2.17	0.7	0.5	0.7	0.17	0.73	0.3
4096	-4.16	2.2	0.5	1.14	0.43	1.07	0.67

Synergy effects: Set2

$reps \| \Sigma \|$	4	8	16	32	64	128	256	512
3	0.0	0.0	0.03	0.0	0.0	0.23	0.1	0.3
4	0.0	0.0	0.0	0.0	0.0	0.27	0.23	0.2
5	0.03	0.0	0.0	0.0	0.04	0.16	0.23	0.23
6	0.0	0.0	0.04	0.0	0.04	0.17	0.5	0.3
7	0.0	0.0	0.0	0.0	0.1	0.17	0.57	1.0
8	0.0	0.0	0.1	0.0	0.0	0.44	0.4	0.1

Relation between LNS and CMSA

First experimental study

Reminder: Intuition

- CMSA will have advantages over LNS when solutions are small, that is, when
  1. solutions consist of few solution components
  2. many variables in the corresponding ILP model have value zero
- LNS will have advantages over CMSA when the opposite is the case

Problem: how to show this?

- Theoretically? hardly possible
- Empirically? Maybe with a parametrizable problem

Example: Multi-dimensional Knapsack Problem (MDKP)

Given:

- A set of items $C = \{1, \ldots, n\}$
- A set of resources $K = \{1, \ldots, m\}$
- Of each resource k we have a maximum quantity c_k (capacity)
- Each item i requires from each resource k a certain quantity $r_{i,k}$
- Each item i has a profit p_i

Valid solutions: Each subset $S \in C$ is a valid solution if

$$\sum_{i \in S} r_{i,k} \leq c_k \quad \forall k \in K$$

Objective function: $f(S) := \sum_{i \in S} p_i$ for all valid solutions S

MDKP: instance tightness

Important parameter: Instance tightness $0 \leq \alpha \leq 1$

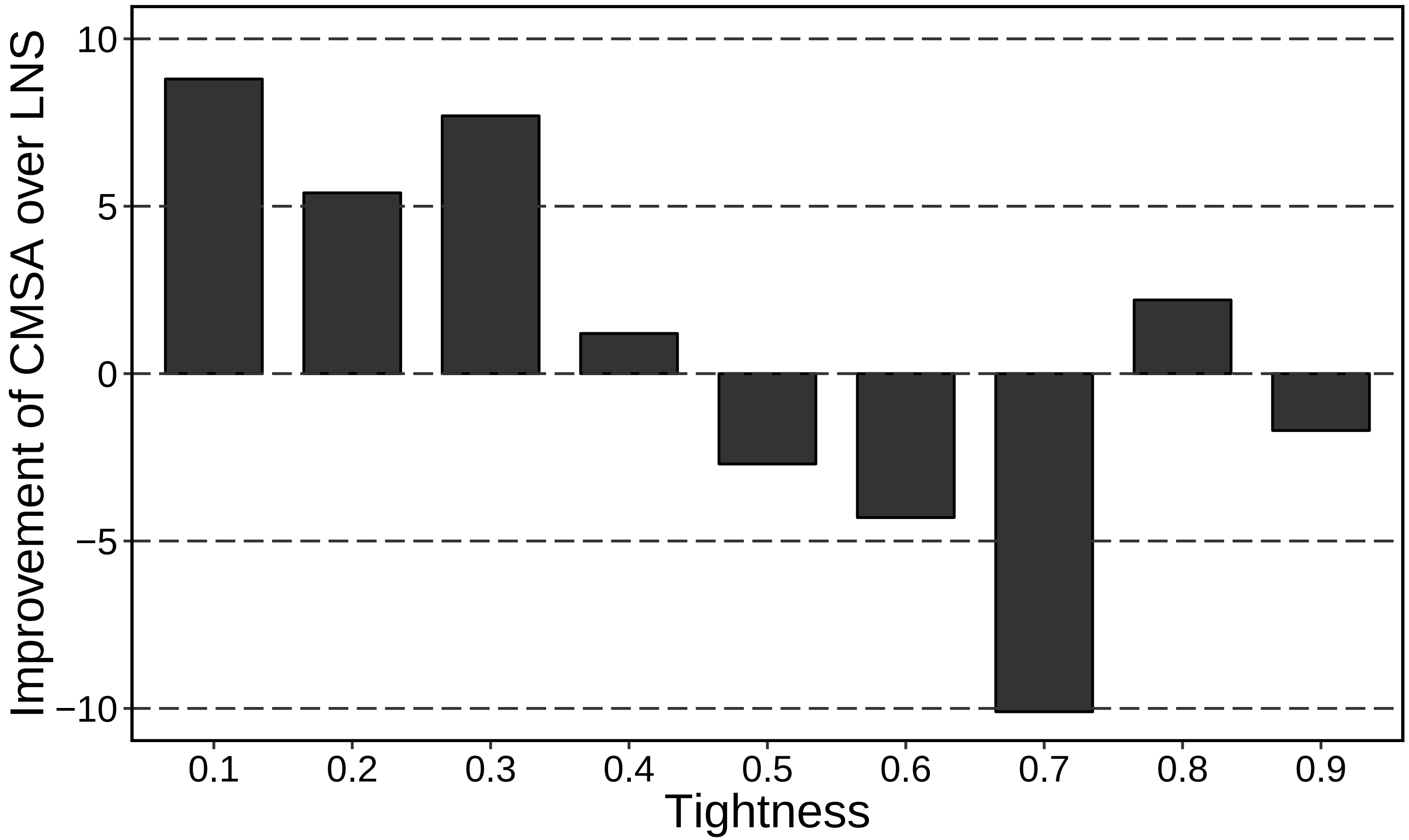
- When α close to zero: capacities are low and valid solution only contain very few items
- When α close to one: capacities are very high and solutions contain nearly all items

Plan:

- Apply both LNS and CMSA to instances from the whole tightness range.
- Both algorithms are tuned with irace seperately for instances of each considered tightness.

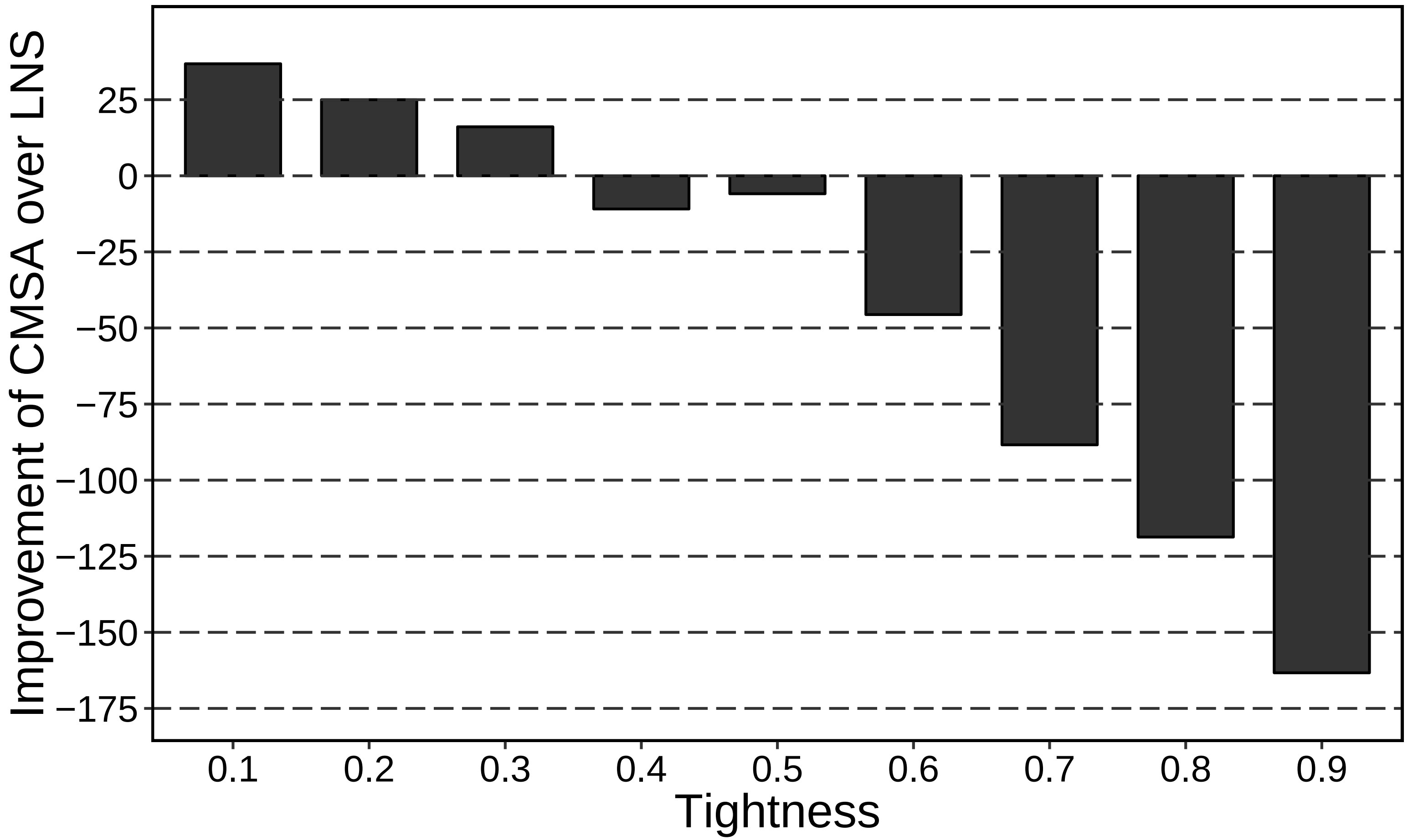
Results for instances with 1000 items

Instance size: $n = 1000$, $m = 10$



Results for instances with 5000 items

Instance size: $n = 5000$, $m = 10$



Summary and Possible Research Directions

Summary:

- Beam-ACO: Hybrid algorithm combining ACO with beam search
- Cmsa: A new matheuristic for combinatorial optimization

Possible Research Directions (CMSA):

- Solution construction: adaptive probabilities over time
- A more intelligent version of the aging mechanism
- Taking profit from research on column generation

People involved in certain aspects of this research

Maria J. Blesa

Borja Calvo

Pedro Pinacho

Evelia Lizárraga

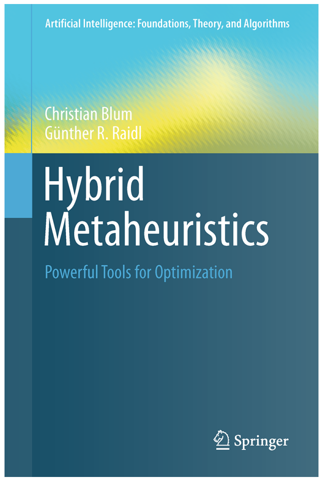
Jóse Antonio Lozano

Manuel López-Ibáñez

Questions?

Literature:

- C. Blum. **Beam-ACO - hybridizing ant colony optimization with beam search: an application to open shop scheduling**, *Computers & OR*, 2005
- C. Blum, P. Pinacho, J. A. Lozano, M. López-Ibáñez. **Construct, Merge, Solve & Adapt: A new general algorithm for combinatorial optimization**. *Computers & Operations Research*, 2016



New book: C. Blum, G. R. Raidl. Hybrid Metaheuristics – Powerful Tools for Optimization, Springer Series on Artificial Intelligence, 2016