

CMSA: A Recent Example of an ILP-based Hybrid Metaheuristic

Christian Blum

Artificial Intelligence Research Institute (IIIA)
Spanish National Research Council (CSIC)

IIIA
Institut d'Investigació en
Intel·ligència Artificial

CSIC
Consejo Superior de Investigaciones Científicas

IIIA-CSIC

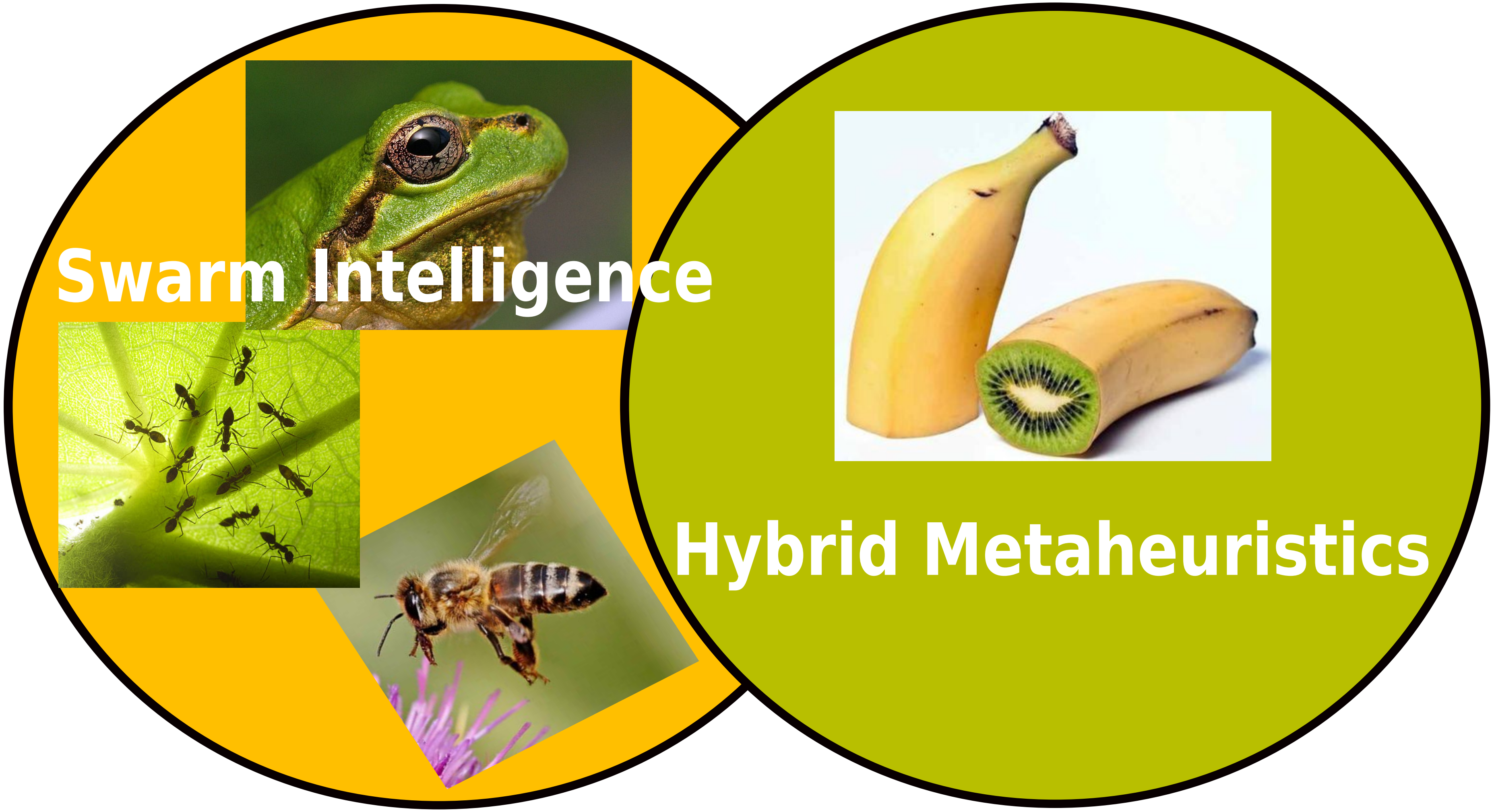
CSIC: Spanish National Research Council

- Largest public institution dedicated to research in Spain (created in 1939)
- Third-largest in Europe
- 6% of all research staff in Spain work for the CSIC
- 20% of the scientific production in Spain is from the CSIC

IIIA: Artificial Intelligence Research Institute

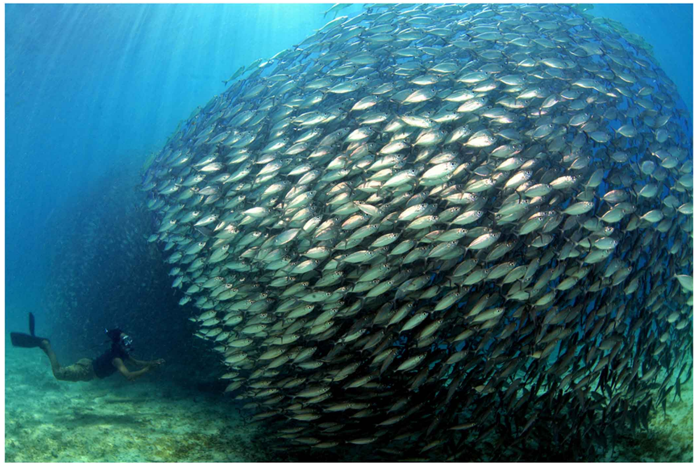
- 18 tenured scientists (of three different ranks)
- Around 30 additional staff member (post-docs, PhD students, technicians, administration)
- Three research lines (machine learning, logic and constraint programming, multi-agent systems)

Research Topics in Recent Years



What is swarm intelligence?

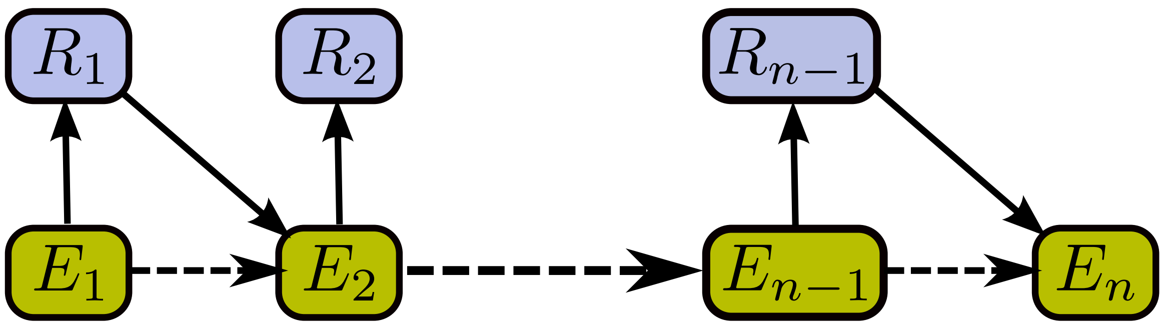
In a nutshell: AI discipline whose goal is designing intelligent multi-agent systems by taking inspiration from the collective behaviour of animal societies such as ant colonies, flocks of birds, or fish schools



Swarm intelligence

Properties:

- Consist of a set of simple entities
- Distributedness: No global control
- Self-organization by:
  - ⋆ **Direct communication:** for example, by visual or chemical contact
  - ⋆ **Indirect communication:** Stigmergy (Grassé, 1959)



Result: Complex tasks/behaviors can be accomplished/exhibited in cooperation

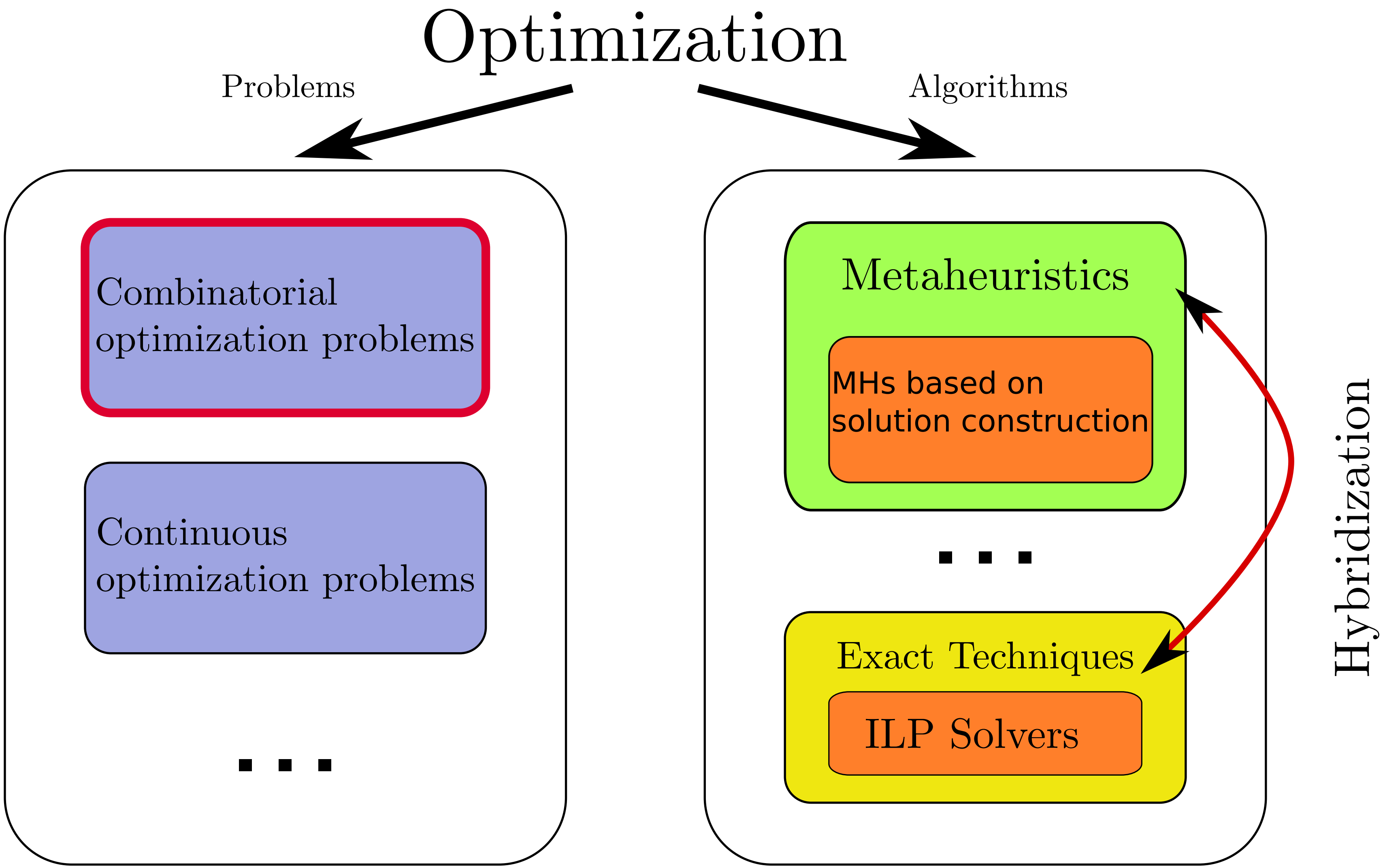
Swarm Intelligence topics from last years

- Combinatorial optimization: ant colony optimization
  Inspiration: foraging behaviour of ant colonies
- Distributed optimization: graph coloring, independent set finding
  Inspiration: self-desynchronization in Japanese tree frogs
- Distributed problem solving: duty-cycling in sensor networks
  Inspiration: work-synchronization in ant colonies

More info: On my website

`https://www.iiia.csic.es/~christian.blum/`

Main Topic of this Presentation: Preparing the Grounds



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 optimization?

- 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 seperate line of research

Motivation behind my work on hybrid metaheuristics

- It is often possible to exploit synergies between different types of algorithms
- 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

Construct, Merge, Solve & Adapt (CMSA)

Short description

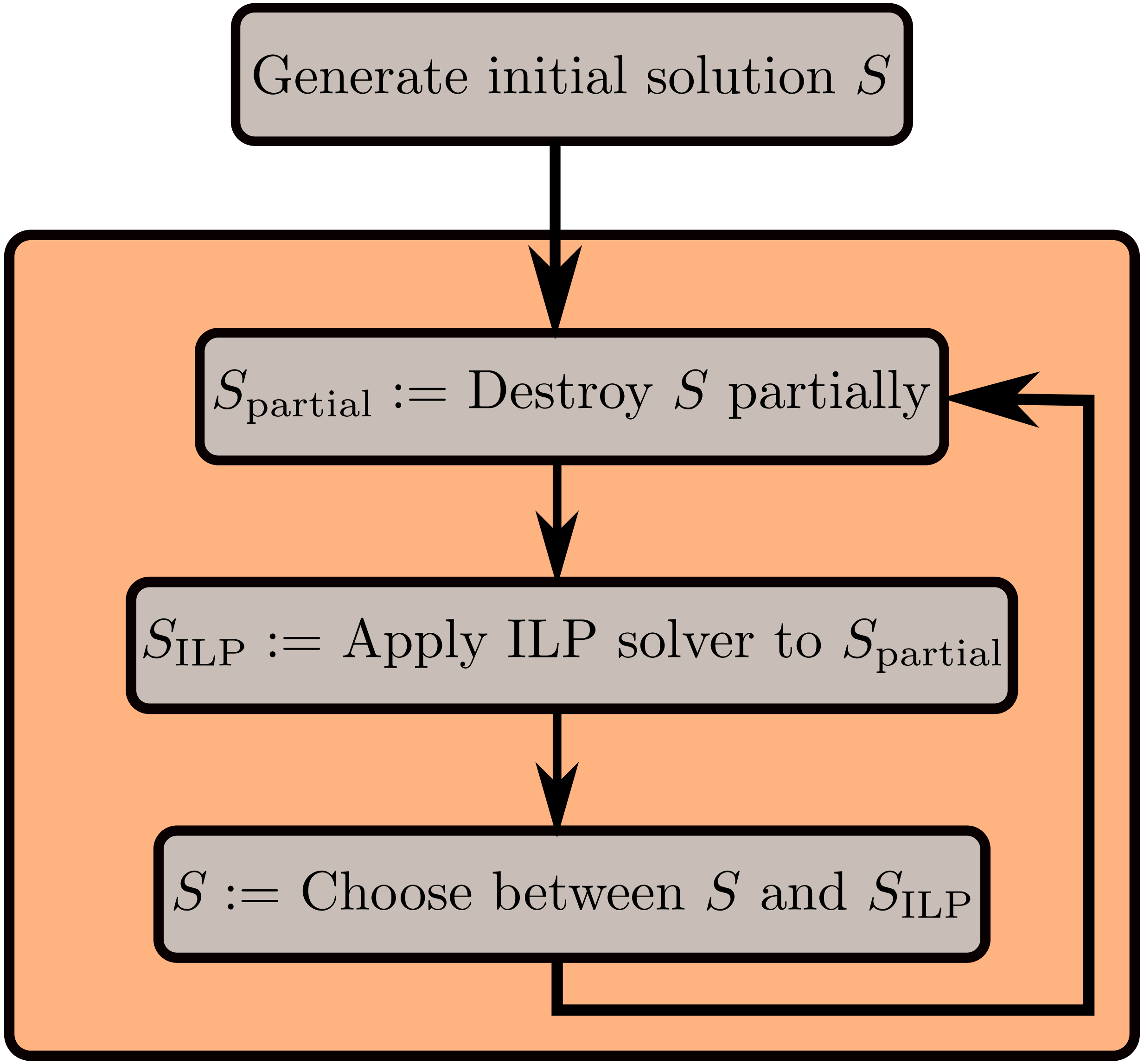
Standard: Large Neighborhood Search

- 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

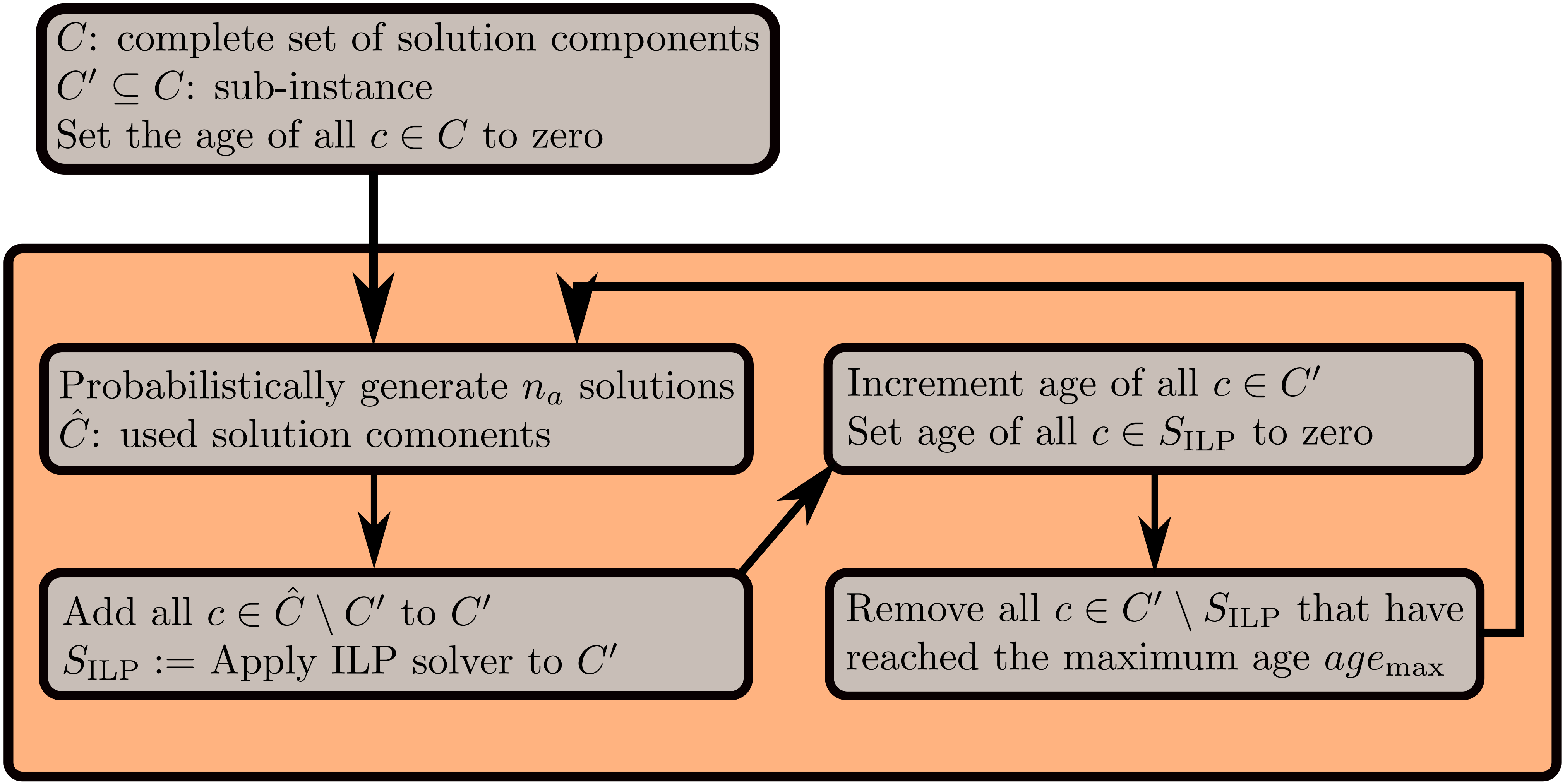
Observation: In the presence of a large number of solutions components, many of them only lead to bad solutions

Idea: Exclude the presumably bad solution components from the ILP

Steps of the proposed method:

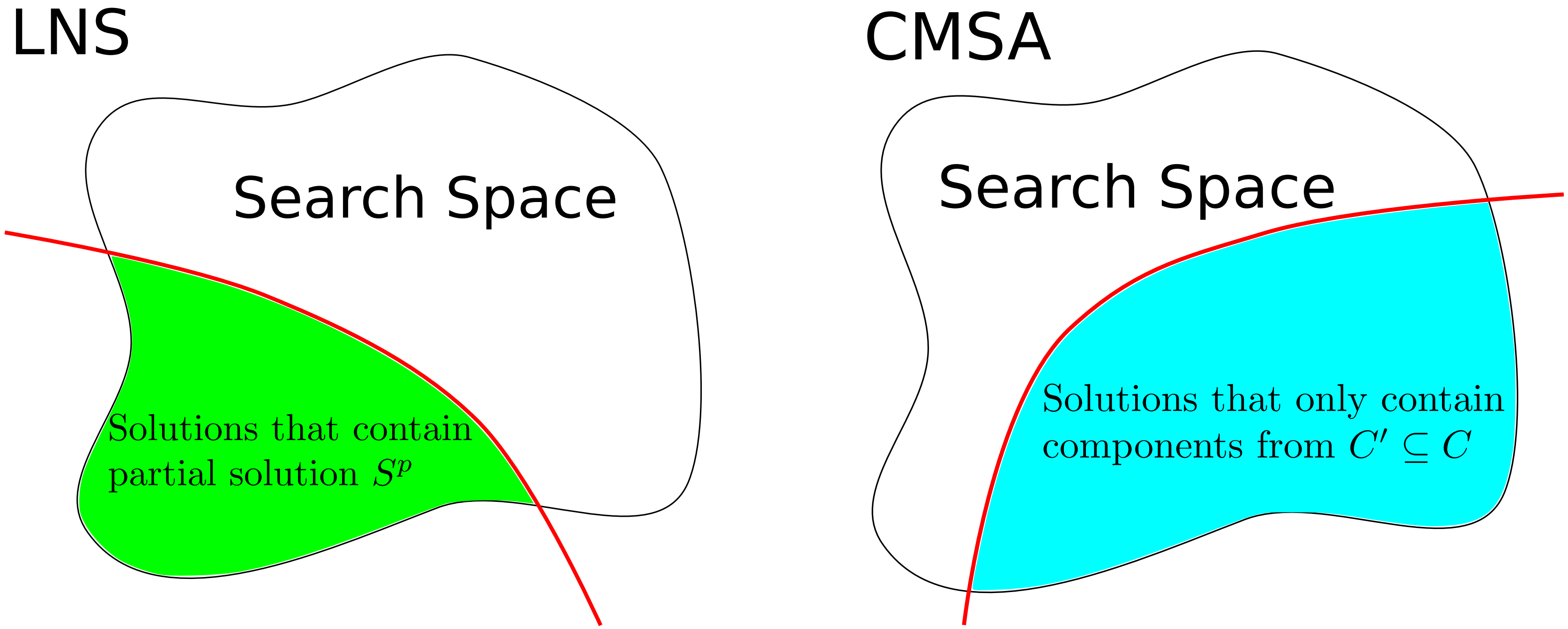
- Iteratively generate presumably good solutions in a probabilistic way
- Assemble a sub-instance from the used solution components
- Solve the sub-instance by means of an ILP solver
- Delete useless solution components from the sub-instance

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

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

Question: How is Σ_t^{nd} defined?

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

$\boxed{\text{Example:}}$ Given is

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

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

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

Greedy function

Greedy function:

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

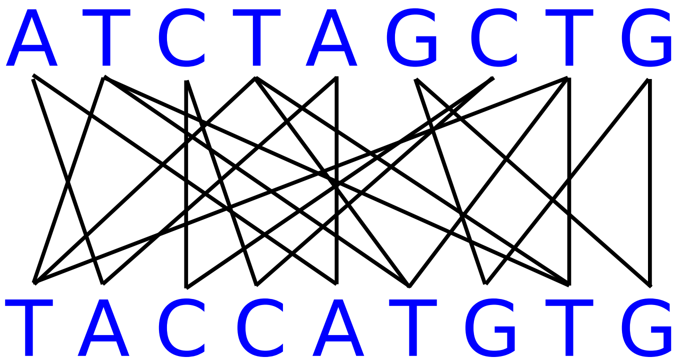
$$\overbrace{\underline{\mathrm{A}}\ \ \mathrm{T}\ \ \underset{\overset{\uparrow}{p_x}}{\underline{\mathrm{C}}}}^{x^+}\quad \overbrace{\underset{\overset{\uparrow}{p_x^{\mathrm{T}}}}{\mathrm{T}}\ \ \mathrm{A}\ \ \underset{\overset{\uparrow}{p_x^{\mathrm{G}}}}{\mathrm{G}}\ \ \mathrm{C}\ \ \mathrm{T}\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

ILP Model (1)

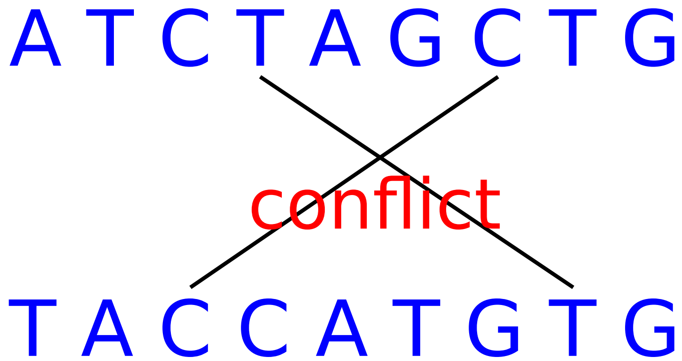
Set of binary variables:

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 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: $|\Sigma| \in \{\mathbf{n/8}, \mathbf{n/4}, \mathbf{3n/8}, \mathbf{n/2}, 5n/8, 3n/4, 7n/8\}$

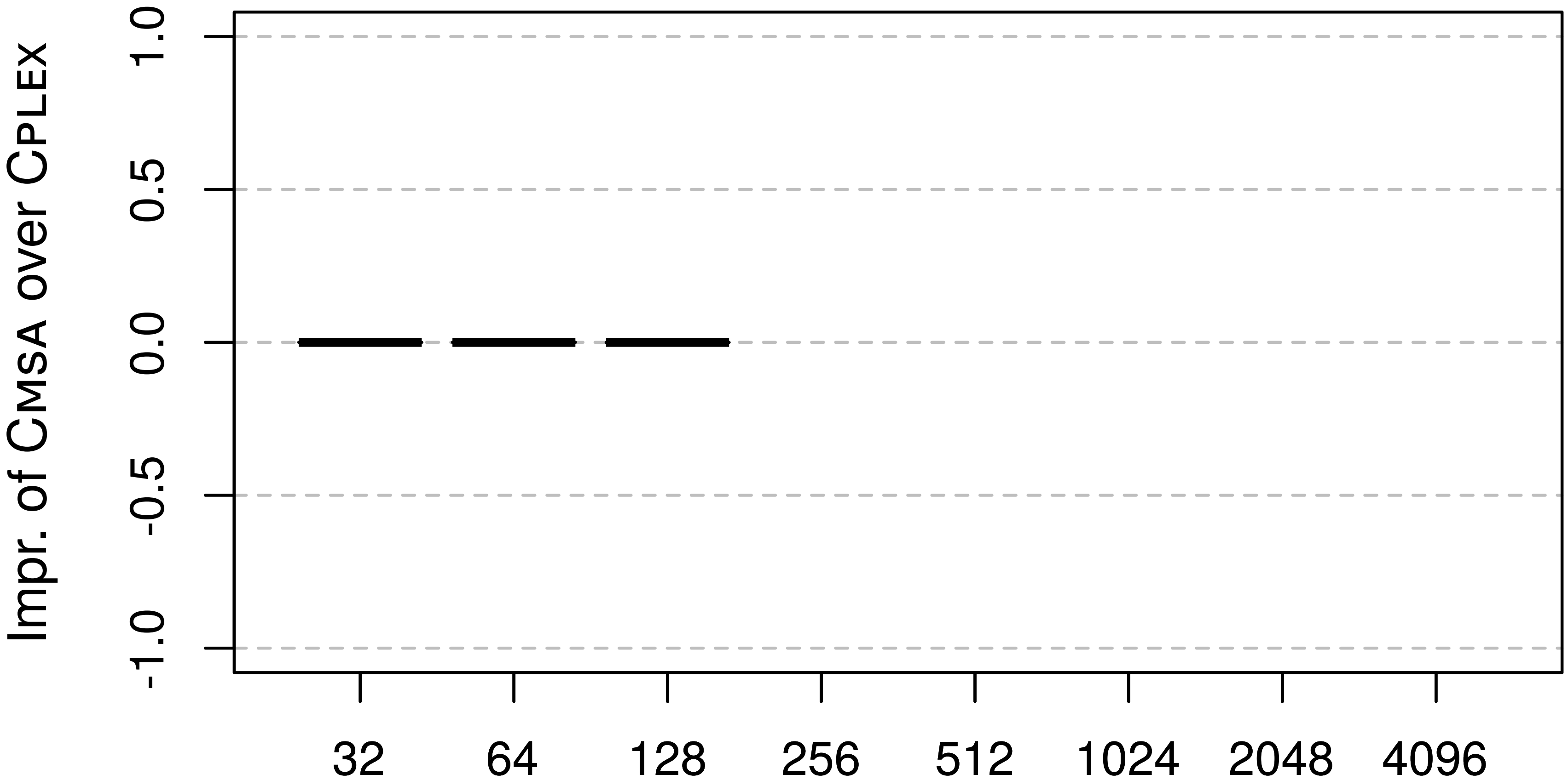
Set2:

- 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

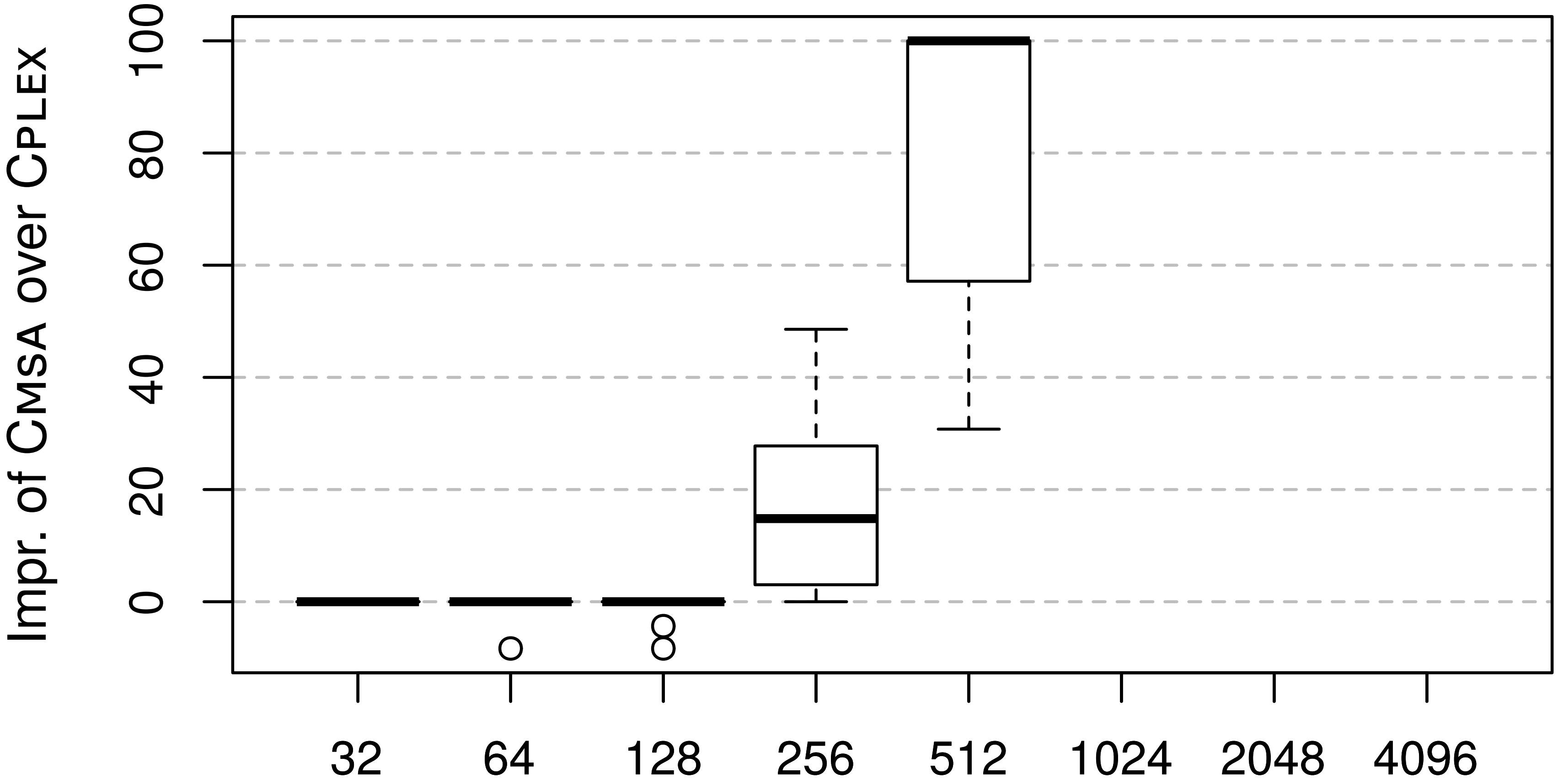
Experimental results: Set1

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



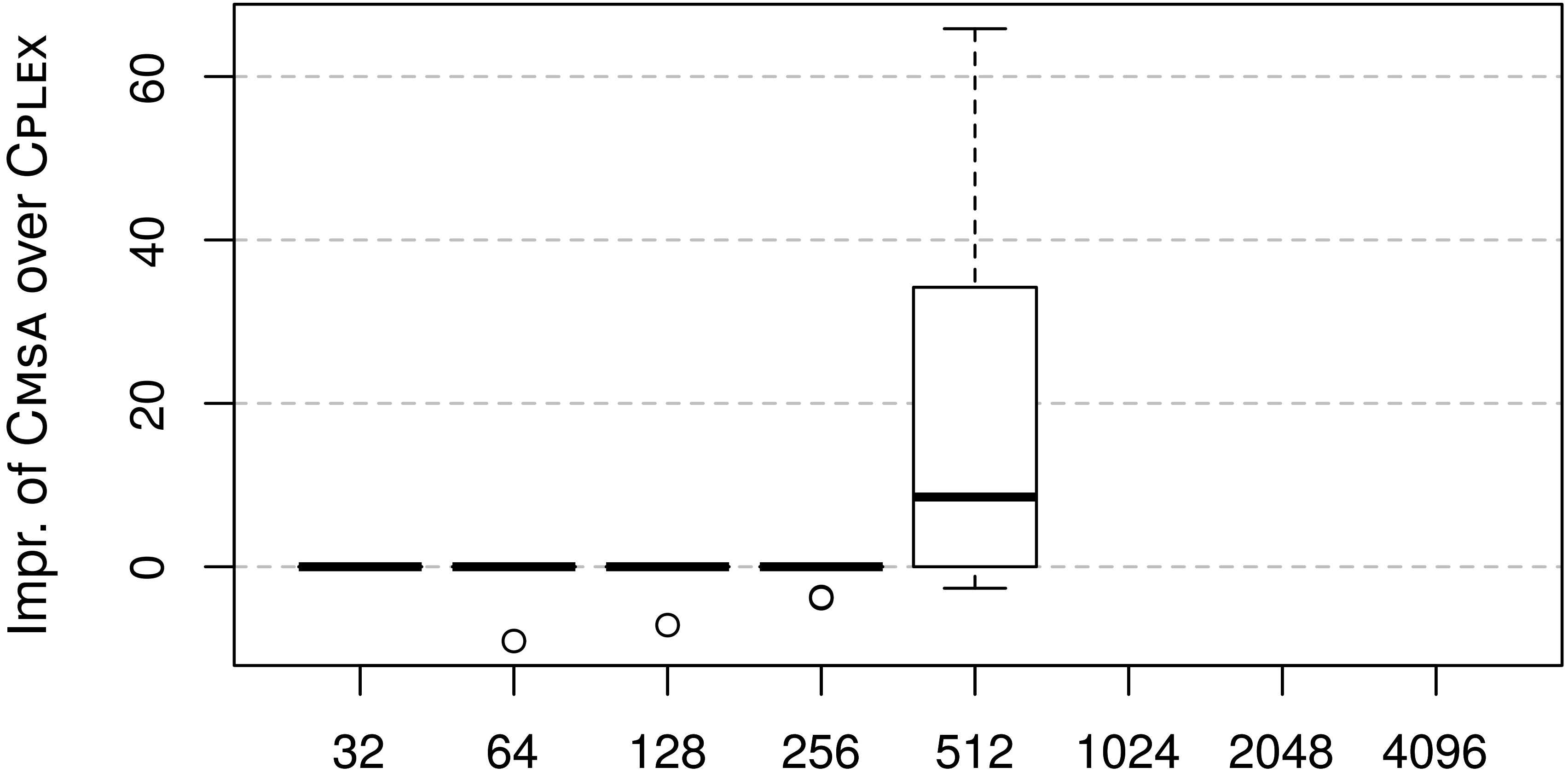
Experimental results: Set1

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



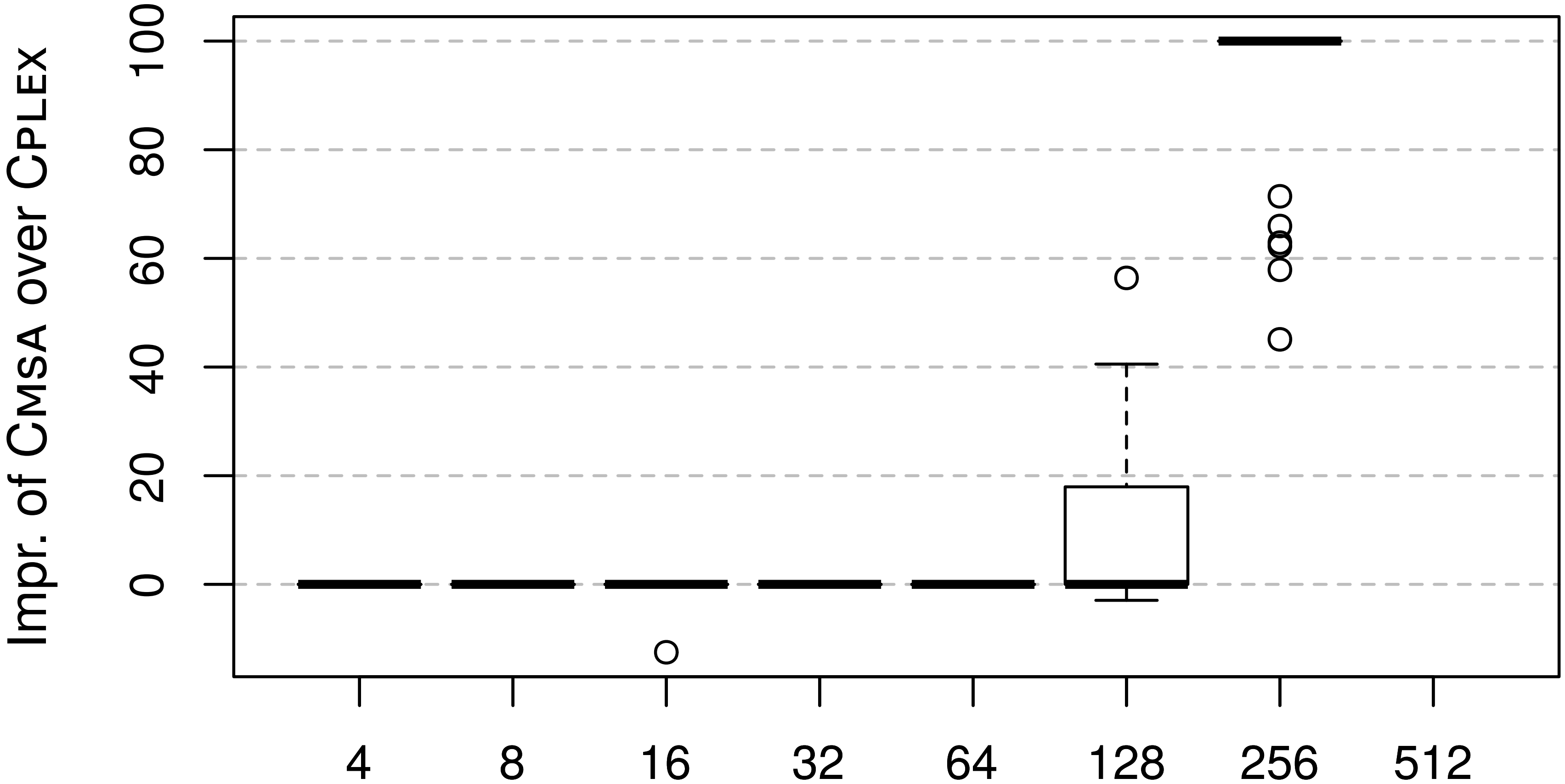
Experimental results: Set1

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



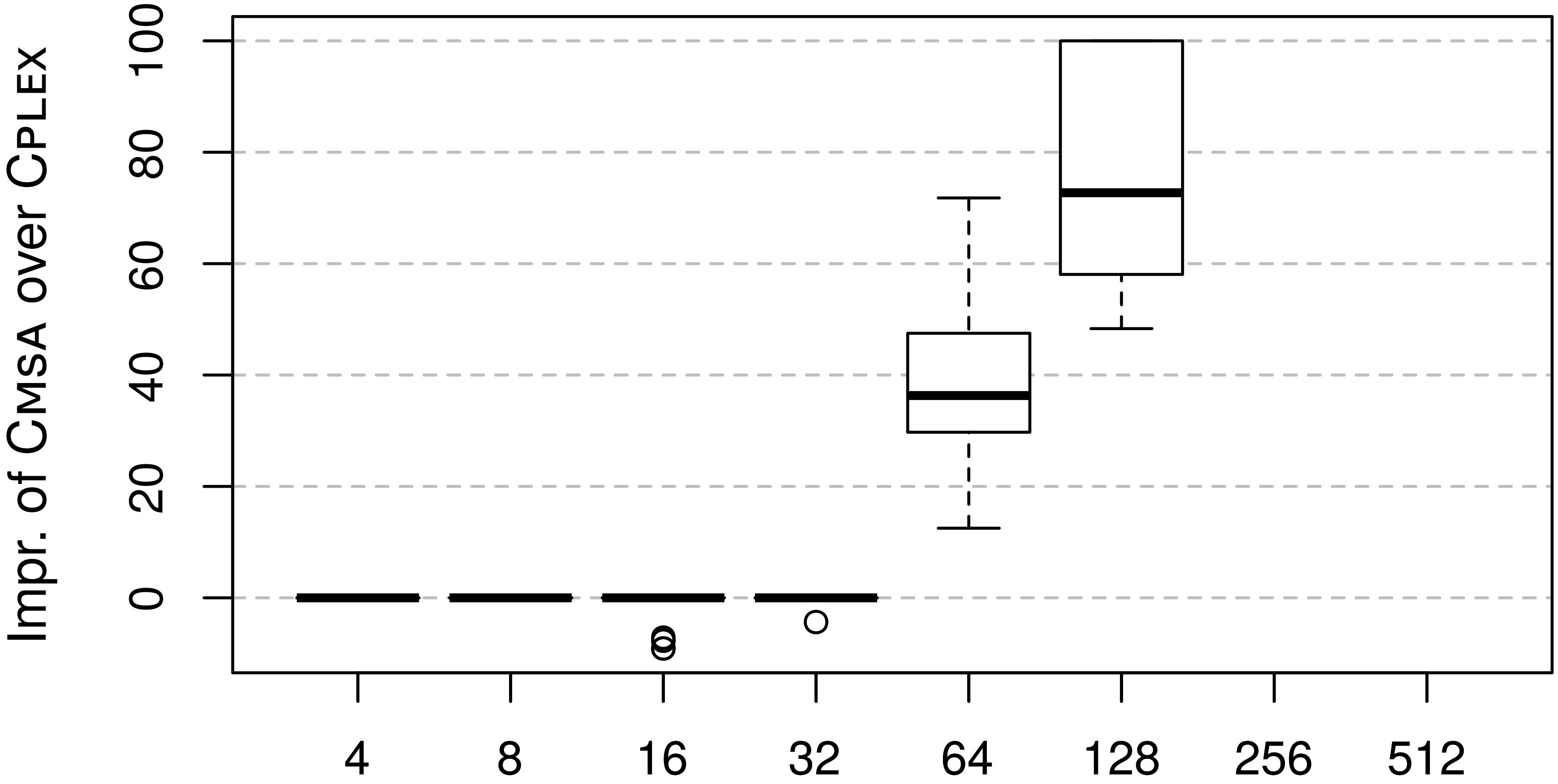
Experimental results: Set2

Improvement of CMSA over CPLEX: 3 reps



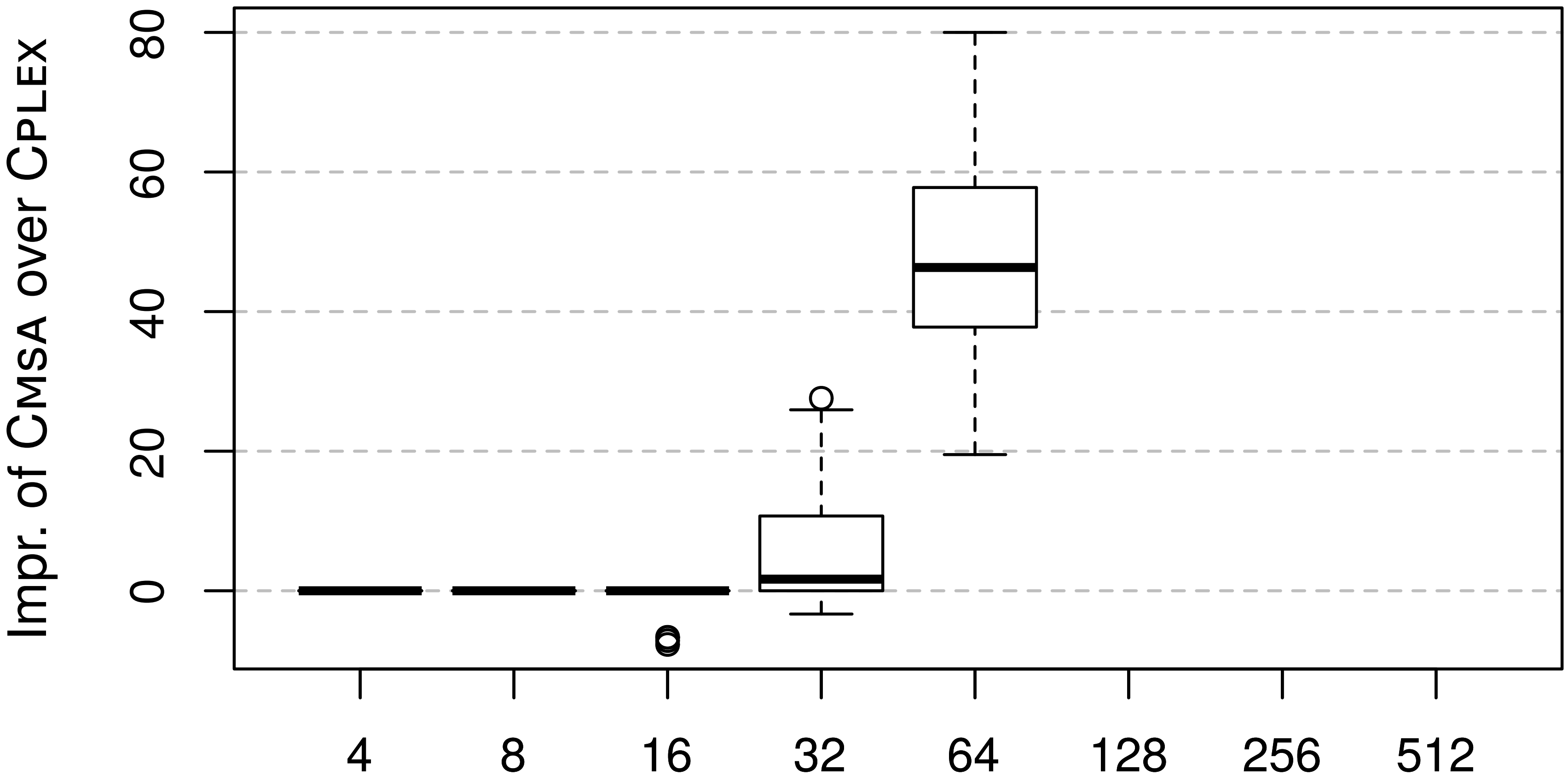
Experimental results: Set2

Improvement of CMSA over CPLEX: 6 reps

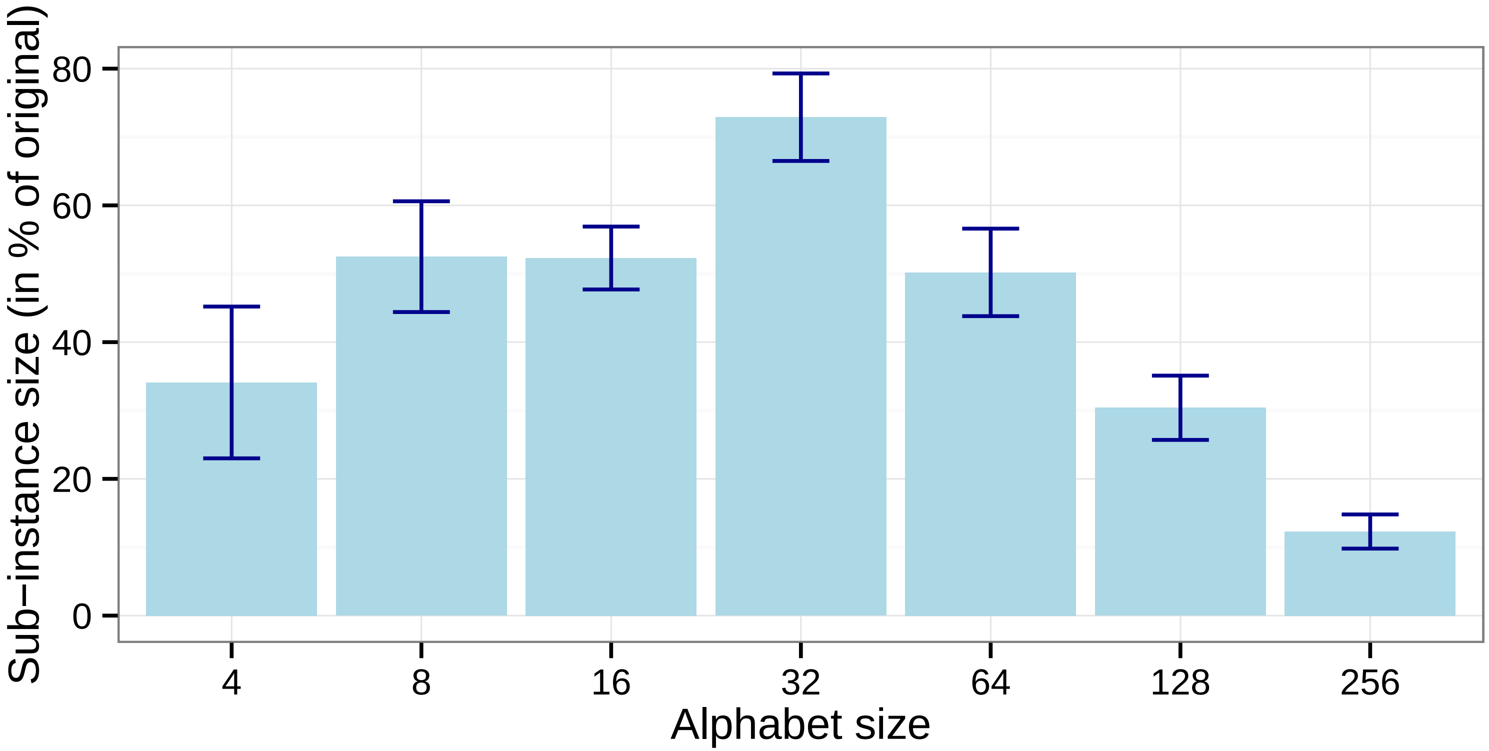


Experimental results: Set2

Improvement of CMSA over CPLEX: 8 reps



Experimental results: size of sub-instances



Relation between LNS and CMSA

An 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

$\boxed{\text{Problem:}}$ 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 $0 \leq \alpha \leq 1$

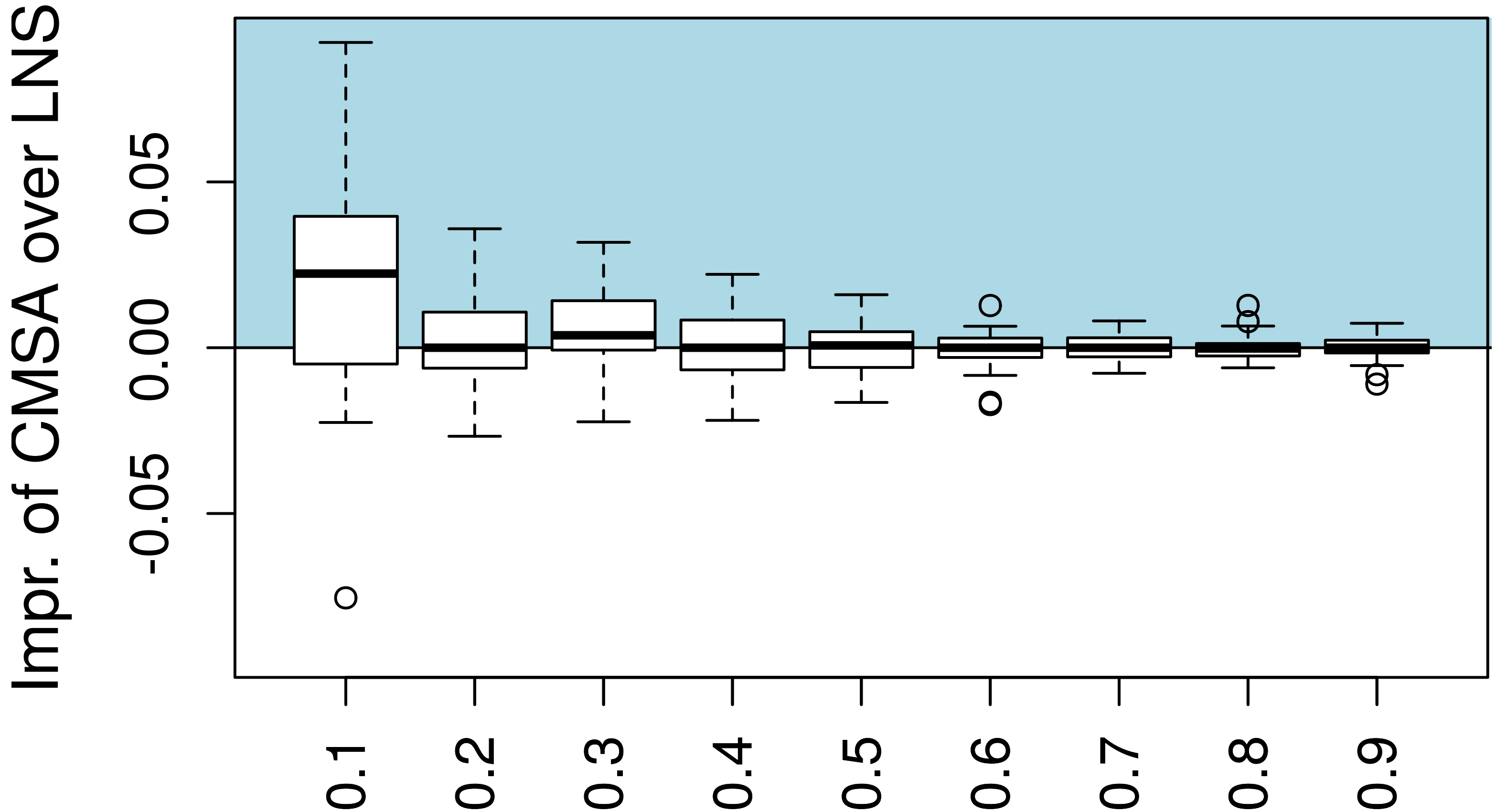
- When α close to zero: capacities are low and valid solution only contain very few items
- When α close to one: capacities are very high and solutions contain nearly all items

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.

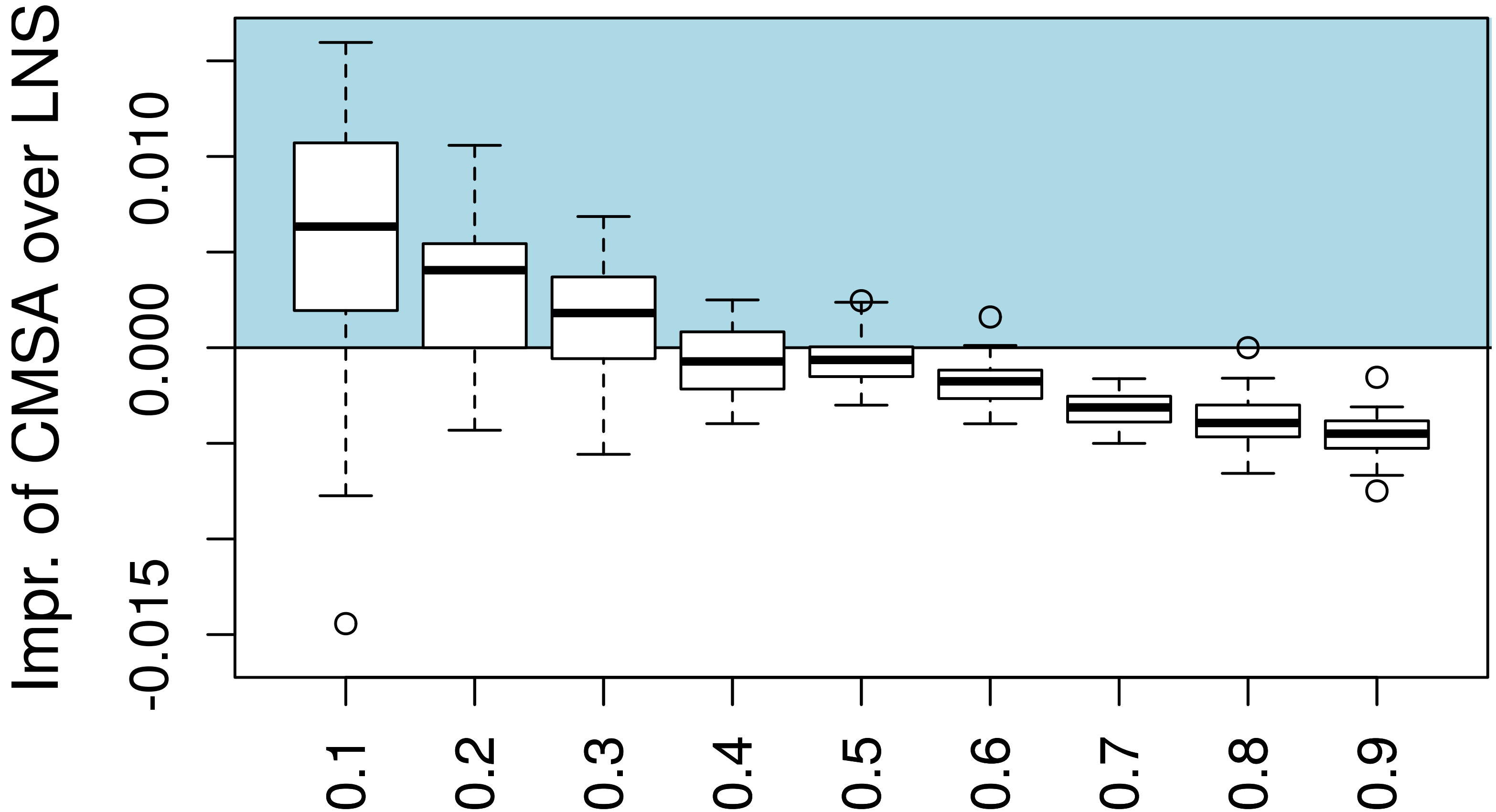
Results for instances with 1000 items

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



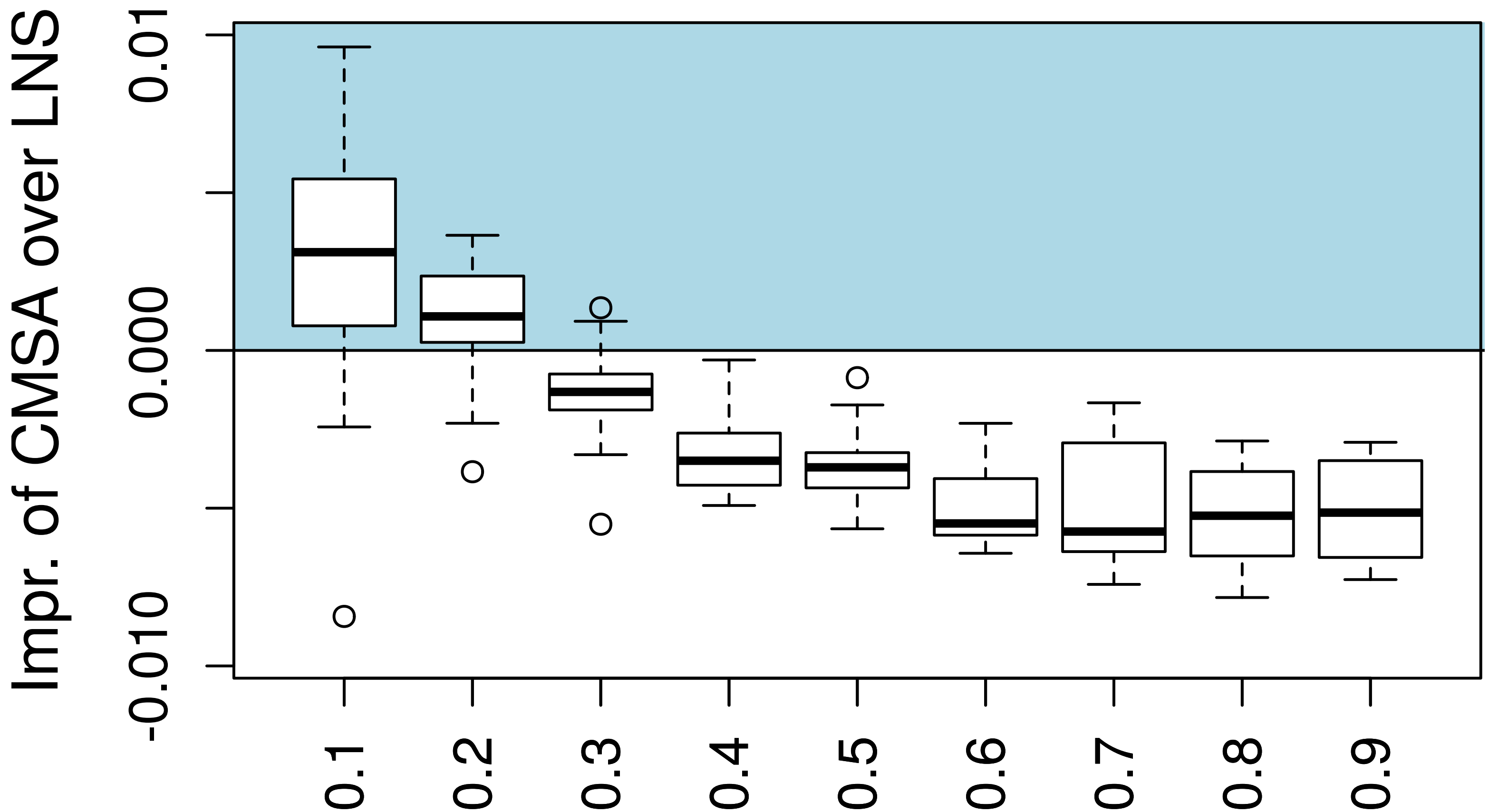
Results for instances with 5000 items

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



Results for instances with 10000 items

Instance size: $n = 10000$, $m = 10$



What if no Efficient Exact Method is Known?

Applying a metaheuristic within CMSA

Weighted Independent Domination Problem: Preliminaries

Given an undirected graph $G = (V, E)$:

- A subset $D \subseteq V$ is called a dominating set if and only if

$$\forall\, v \in V \text{ it holds that } N[v] \cap D \neq \emptyset$$

- A subset $I \in V$ is called an independent set if and only if no two vertices from I are adjacent in G

NP-hard problems:

- Minimum Dominating Set problem
- Maximum Independent Set problem

Weighted Independent Domination Problem (WIDP)

Given:

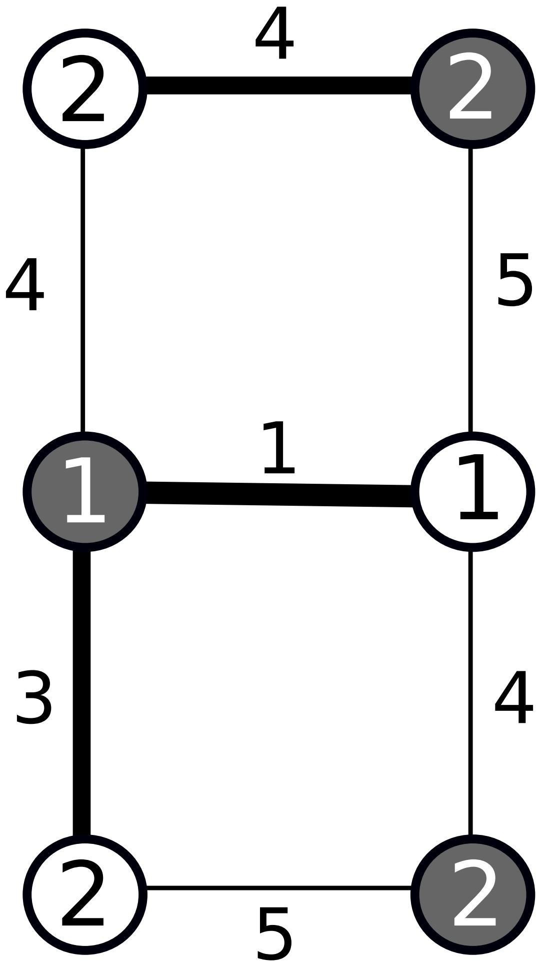
- An undirected graph $G = (V, E)$
- Each node $v \in V$ has an integer weight $w(v) \geq 0$
- Each edge $e \in E$ has an integer weight $w(e) \geq 0$

Valid solutions: Any set $D \subseteq V$ which is at the same time a dominating set and an independent set

Objective function: given a valid solution $D \subseteq V$

$$f(D) := \sum_{u \in D} w(u) + \sum_{v \in V \setminus D} \min\{w(e = (v, u)) \mid u \in \{N(v) \mid v \in D\}\}$$

The WIDP Problem: An Example



- Complexity: shown to be NP-hard
- Only algorithmic approach: A linear time algorithm for series parallel graphs

Integer Linear Programming (ILP) Model

$$
\begin{aligned}
\text{(ILP)} \quad \min \quad & \sum_{v \in V} x_v w(v) + \sum_{e \in E} z_e w(e) && && (5) \\
\text{s.t.} \quad & x_v + x_u \leq 1 && \text{for } e = (u,v) \in E && (6) \\
& x_v + x_u = y_e && \text{for } e = (u,v) \in E && (7) \\
& z_e \leq y_e && \text{for } e \in E && (8) \\
& x_v + \sum_{u \in N(v)} x_u \geq 1 && \text{for } v \in V && (9) \\
& x_v + \sum_{e \in \delta(v)} z_e \geq 1 && \text{for } v \in V && (10) \\
& x_v \in \{0,1\} && \text{for } v \in V && \\
& y_e \in \{0,1\} && \text{for } e \in E && \\
& z_e \in \{0,1\} && \text{for } e \in E &&
\end{aligned}
$$

Greedy Heuristics: General Structure

input: a undirected graph $G = (V, E)$ with node and edge weights
$S := \emptyset$
$G' := G$
while $V' \neq \emptyset$ **do**
    $v^* :=$ ChooseFrom(V')
    $S := S \cup \{v^*\}$
    Remove from G' all nodes from $\{v^*\} \cup N(v^* \mid G')$ (and the incident edges)
end while
output: An independent dominating set S of G

Note:

- G' is the graph that remains when removing nodes (and the incident edges)

Greedy Heuristics: How To Choose a Node?

Greedy1: Implementation of ChooseFrom(V')

$$v^* := \operatorname{argmax} \left\{ \frac{|N(v \mid G')|}{w(v)} \mid v \in V' \right\} \tag{11}$$

Note:

- This heuristic favors nodes with a high remaining degree and a low node weight
- Edge weights are not considered, only at the time of computing the objective function value

Greedy Heuristics: How To Choose a Node?

GREEDY2: Given a partial solution $S \subset V$, the contribution $c(v \mid S)$ of a node $v \in V'$ is defined as follows:

1. If $v \in S$: $c(v \mid S) := w(v)$
2. If $v \notin S$ and $N(v) \cap S = \emptyset$: $c(v \mid S) := \max\{w(e) \mid e \in E\}$
3. If $v \notin S$ and $N(v) \cap S \neq \emptyset$: $c(v \mid S) := \min\{w(e) \mid e = (v, u), u \in S\}$

Implementation of ChooseFrom(V'): Via aux. func. $f^{\text{aux}}(S) := \sum_{v \in V} c(v \mid S)$

$$v^* := \operatorname{argmin}\left\{f^{\text{aux}}(S \cup \{v\}) \mid v \in V'\right\}$$

Population-Based Iterated Greedy (PBIG)

input: input graph G, parameters $p_{\text{size}} > 0$, $D^l, D^u, d_{\text{rate}}, l_{\text{size}} \in [0,1]$
$\mathcal{P} :=$ GenerateInitialPopulation$(p_{\text{size}}, d_{\text{rate}}, l_{\text{size}})$
while termination condition not satisfied **do**
  $\mathcal{P}_{\text{new}} := \emptyset$
  for each candidate solution $S \in \mathcal{P}$ **do**
    $\hat{S} :=$ DestroyPartially(S)
    $S' :=$ Reconstruct$(\hat{S}, d_{\text{rate}}, l_{\text{size}})$
    AdaptDestructionRate(S, S')
    $\mathcal{P}_{\text{new}} := \mathcal{P}_{\text{new}} \cup \{S'\}$
  end for
  $\mathcal{P} :=$ Best p_{size} solutions from $\mathcal{P} \cup \mathcal{P}_{\text{new}}$
end while
output: argmin $\{f(S) \mid S \in \mathcal{P}\}$

PBIG: Probabilistic (Re-)Construction of Solutions

Characteristics:

- Uses mechanism and greedy function of Greedy2.
- Makes use of a determinism rate d_{rate} and a candidate list size l_{size}
- **At each construction step:**
  - ⋆ First draw a random number $\delta \in [0, 1]$.
  - ⋆ If $\delta \leq d_{\text{rate}}$: Choose the best node from $v \in V'$
  - ⋆ If $\delta > d_{\text{rate}}$: Choose randomly from the best l_{size} nodes

PBIG: Partial Destruction of Solutions

Mechanism:

- Randomly remove D_r percent of all nodes from a solution
- Dynamic solution-specific destruction rate D_r

Parameters: $0 \leq D^l \leq D^u \leq 1$

- D^l: minimum destruction rate
- D^u: maximum destruction rate
- D^{inc}: increment of the destruction rate (fixed to 0.05)

Management of D_r:

- Start with $D_r := D^l$. Moreover, whenever a better solution is found set D_r back to D^l
- When no better solution is found: $D_r := D_r + D^{\text{inc}}$

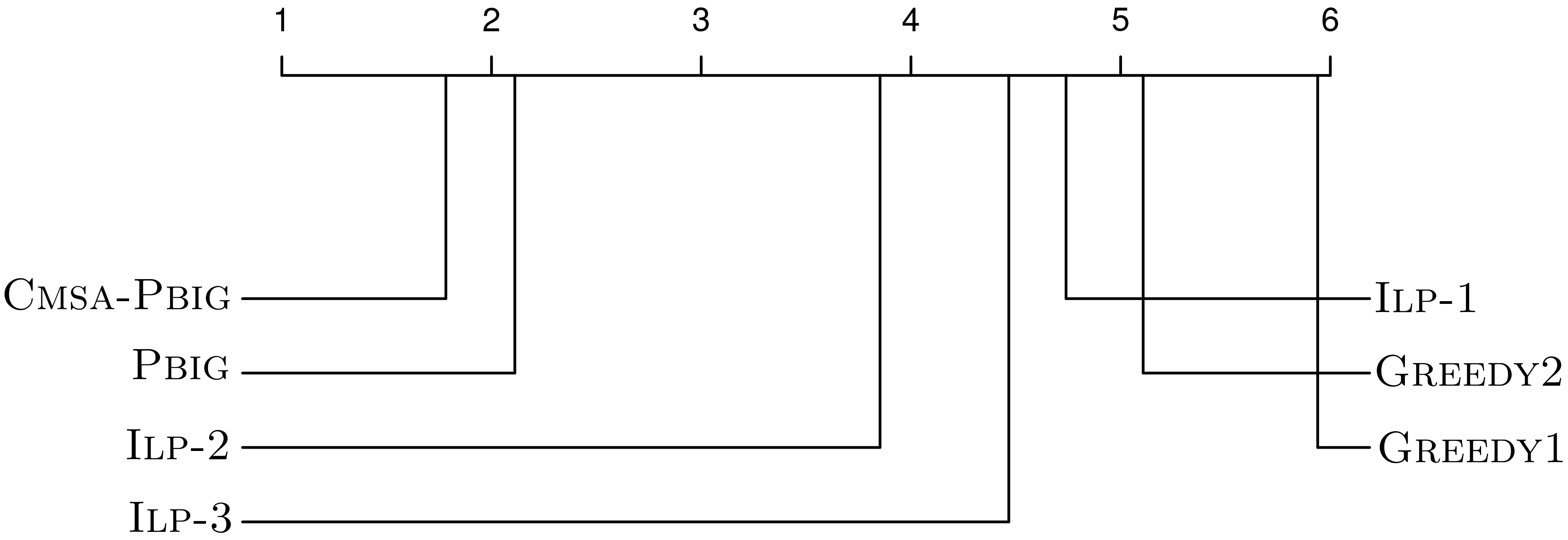
Experimental Evaluation: Instances

Considered graphs: random graphs and random geometric graphs

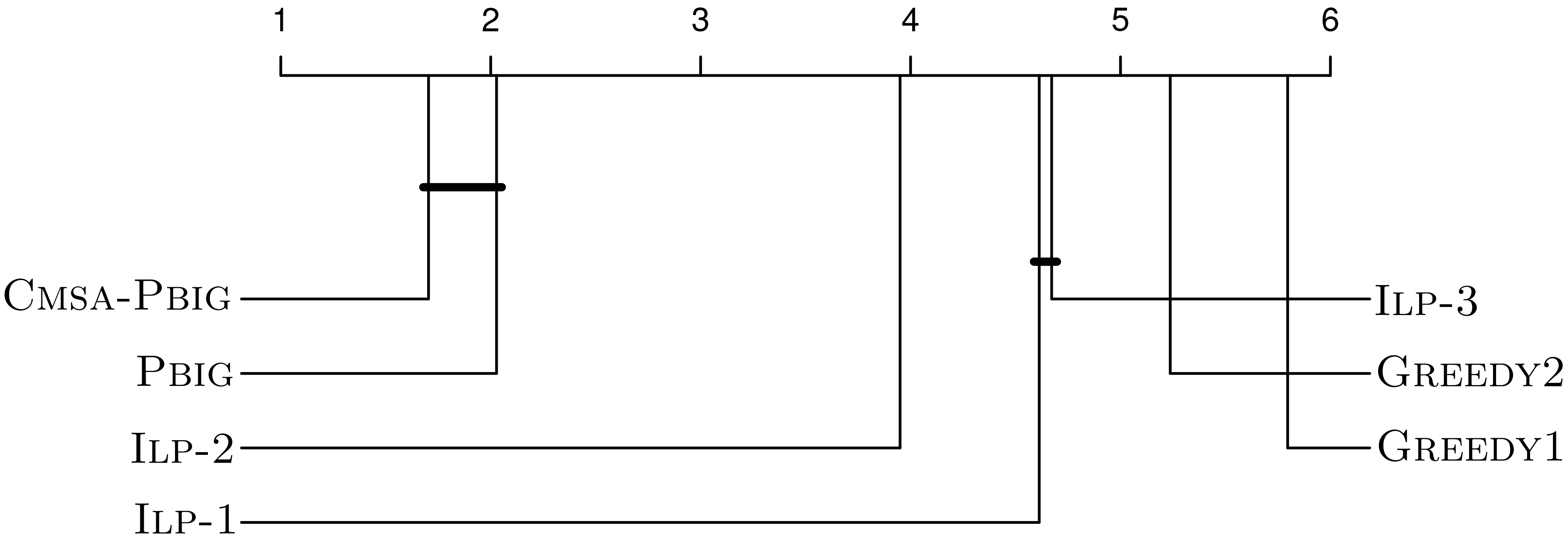
- $|V| \in \{100, 500, 1000\}$
- Random graphs: edge probability $ep \in \{0.05, 0.15, 0.25\}$
- Random geom. graphs: radius $r \in \{0.14, 0.24, 0.34\}$
- **Neutral graphs** node/edge weights uniformly at random from $\{0, \ldots, 100\}$
- **Node-oriented graphs** node weights from $\{0, \ldots, 1000\}$, edge weights from $\{0, \ldots, 10\}$
- **Edge-oriented graphs** node weights from $\{0, \ldots, 10\}$, node weights from $\{0, \ldots, 1000\}$

Number of graphs: 10 graphs for each comb. of $|V|$, ep/r and graph type (a total of 540 graphs)

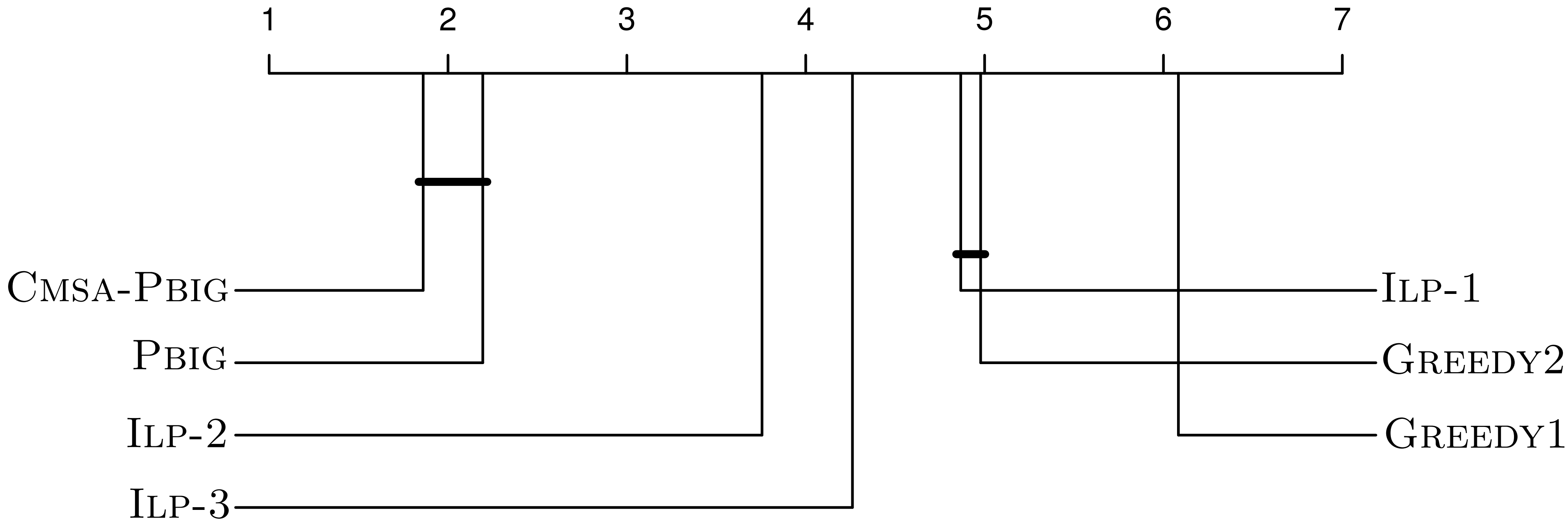
Critical Difference Plots: Global Picture



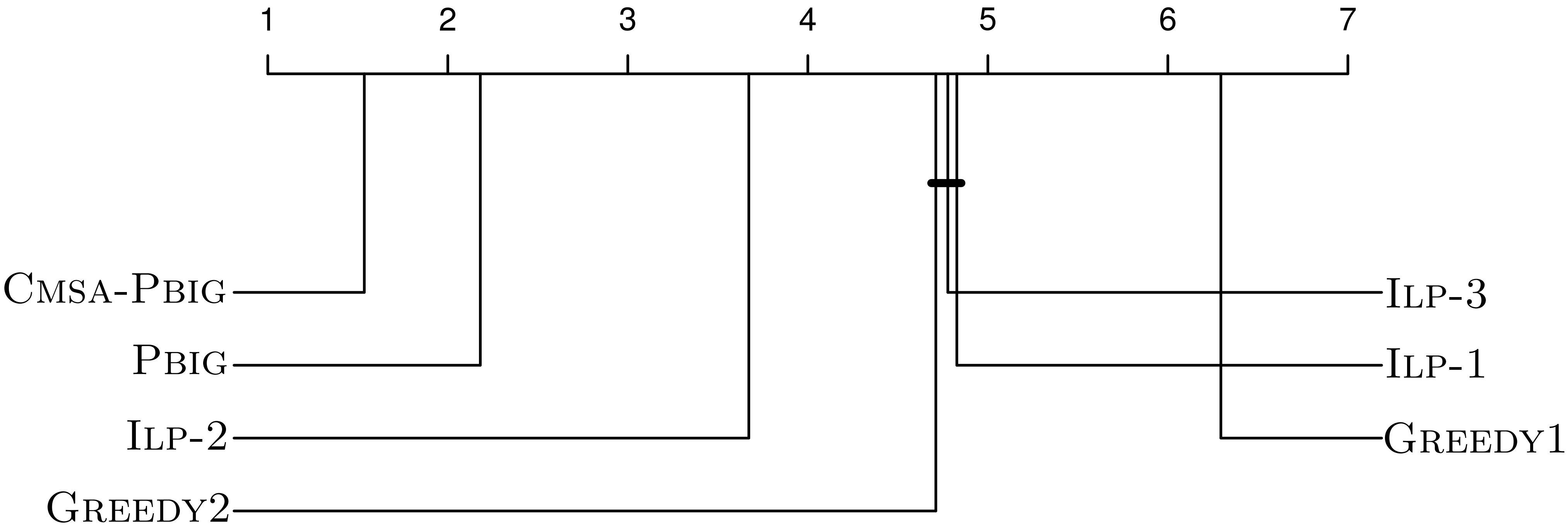
Critical Difference Plots: Random Graphs



Critical Difference Plots: Random Geometric Graphs



Critical Difference Plots: Edge-Oriented Graphs



Summary and Possible Research Directions

Summary:

- **CMSA:** A new hybrid metaheuristic for combinatorial optimization
- **Goal:** Make ILP solvers applicable to larg(er) problem instances

Possible Research Directions:

- **Solution construction:** adaptive probabilities over time
- A more intelligent version of the **aging mechanism**
- Identify applications where **constraint programming** can be useful as exact technique inside CMSA

Thanks to colleagues involved in this research

Pedro Pinacho

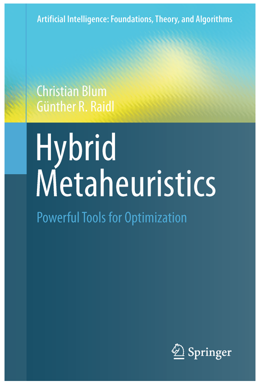
Jóse Antonio Lozano

Manuel López-Ibáñez

Questions?

Literature:

- 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
- P. Pinacho, C. Blum, J. A. Lozano. **The Weighted Independent Domination Problem: Integer Linear Programming Models and Metaheuristic Approaches**. *European Journal of Operational Research*, in press (2017)



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