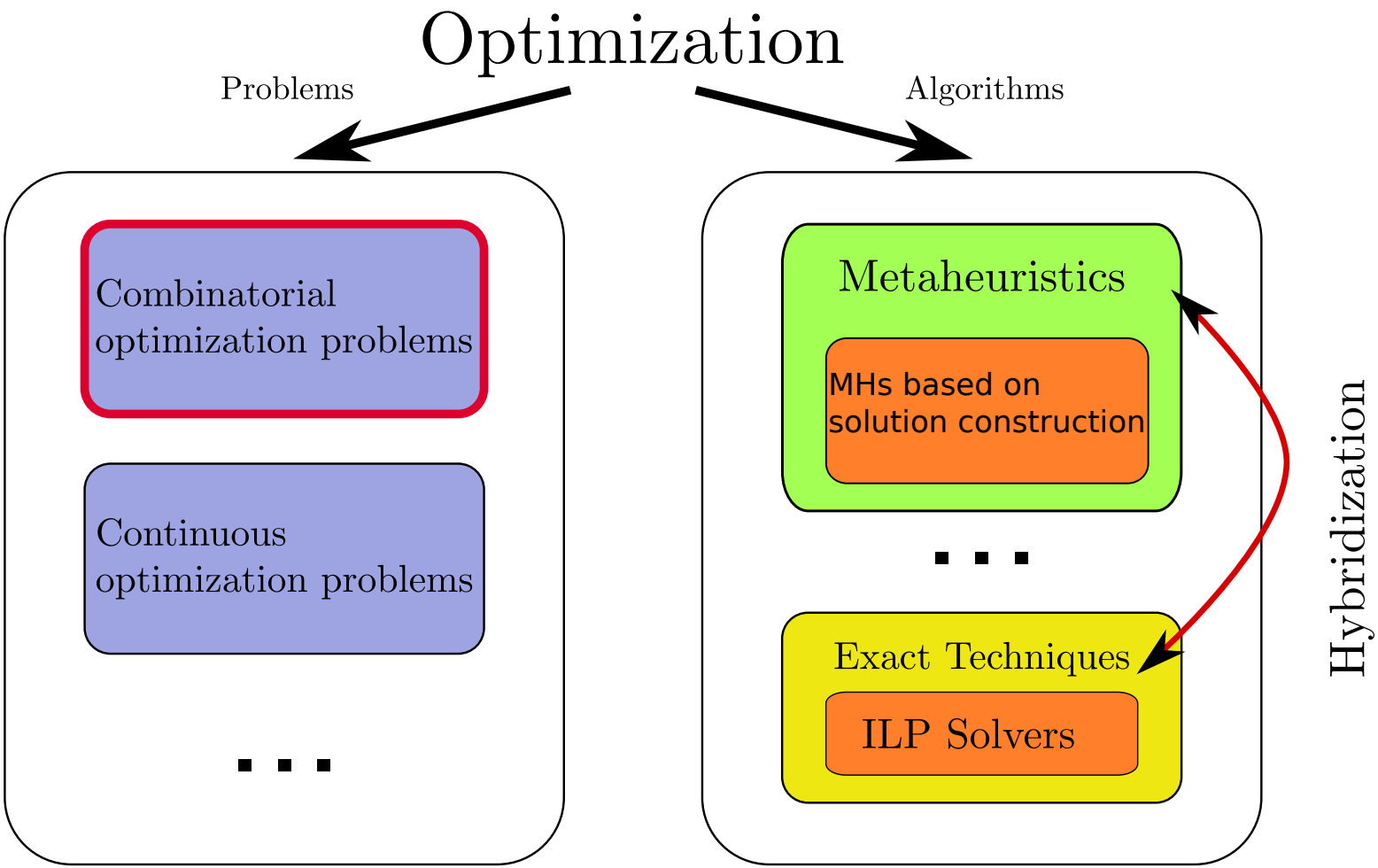
Hybrid Metaheuristics (Vienna 2022)

©Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)

Hybrid metaheuristics



Exact methods

Exact methods for combinatorial optimization

- **Exhaustive search** (not practical)
- **Mathematical programming**
  - Branch & bound (B&B)
  - Branch & cut (B&B + *cutting planes* to tighten LP relaxations)
  - Branch & price (hybrid between B&B and *column generation*)
- **Dynamic programming**
- **Constraint programming**

Generic branch & bound

B&B, general description. Clarification on the board!

- Branch & Bound (B&B) is a tree search algorithm
- Each node of the search tree corresponds to a sub-problem
- B&B make use of a strategy for the sub-division of sub-problems (*branching rule*)
- B&B uses bounding information in order to discard sub-problems and prove optimality
- Optionally uses a heuristic

Generic branch & bound

Pseudo code

1: $B := \infty$
2: **if** heuristic available **then** $s :=$ ApplyHeuristic(); $B := f(s)$; $s_{\text{bsf}} := s$
3: Initialize a queue Q containing only the root node of the search tree
4: **while** Q is not empty **do**
5: Choose a node N from the queue Q (and remove N from Q)
6: **if** N contains a single solution s' **then**
7: **if** $f(s') < B$ **then** $B := f(s')$ and $s_{\text{bsf}} := s'$ **end if**
8: **else**
9: Use the branching rule to produce a set of sub-problems N_{sub} from N
10: **for** all $N_i \in N_{\text{sub}}$ **do**
11: **if** $LB(N_i) \leq B$ **then** add N_i to Q **end if**
12: **end for**
13: **end if**
14: **end while**

Generic branch & bound

Different ways to handle queue Q

- **Last-in-first-out** (LIFO): results in a **depth-first** algorithm
- **First-in-first-out** (FIFO): results in a **breath-first** algorithm
- **When the ordering in queue Q depends on $LB()$:** results in a **best-first** algorithm

Note

When no heuristic is available, the FIFO variant is preferable.

Dynamic programming (DP)

How does it work?

1. Divide the given problem into sub-problems
2. Combine solutions of already solved sub-problems to solutions to bigger sub-problems until a solution for the original problem is obtained

Required properties of the problem

1. **Optimal substructure:** optimal solution to the problem must contain optimal solutions to sub-problems
2. **Overlapping sub-problems:** solutions to high-level sub-problems reuse lower level sub-problems
3. **Size of the sub-problem space:** should be of moderate size (polynomial)

DP example: LCS problem

Definition: *longest common subsequence* (LCS) problem

- **Given:** input strings x and y of length $|x| = n$ y $|y| = m$
- $x = x[1] \dots x[n]$, $y = y[1] \dots y[m]$
- Let x_i denote the prefix of x until position i
- **Example:** if $x =$ **BANANA** then $x_3 =$ **BAN**
- Let $LCS(x, y)$ denote the LCS between x and y

Note

1. Case 1: $x[n] = y[m]$ $LCS(x, y) = LCS(x_{n-1}, y_{m-1}) + x[n]$
2. Case 2: $x[n] \neq y[m]$
   - Sub-case 2.1: if $LCS(x, y)$ ends with $x[n]$.
     Then: $LCS(x, y) = LCS(x, y_{m-1})$
   - Sub-case 2.2: if $LCS(x, y)$ does not end with $x[n]$.
     Then: $LCS(x, y) = LCS(x_{n-1}, y_m)$

DP example: LCS problem

Resulting DP formula

For any $i \in \{0, 1, \ldots, n\}$ and $j \in \{0, 1, \ldots, m\}$

$$LCS(x_i, y_j) = \begin{cases} \epsilon \quad \text{(empty string)} & \text{if } i = 0 \text{ or } j = 0 \\ LCS(x_{i-1}, y_{j-1}) + x[i] & \text{if } x[i] = y[j] \\ \max(LCS(x_i, y_{j-1}), LCS(x_{i-1}, y_j)) & \text{if } x[i] \neq y[j] \end{cases}$$

DP example: LCS problem

Application of the DP formula

		A	B	C	D	A
	0	0	0	0	0	0
A	0	1	1	1	1	1
C	0	1	1	2	2	2
B	0	1	2	2	2	2
D	0	1	2	2	3	3
E	0	1	2	2	3	3
A	0	1	2	2	3	4

LCS - "ACDA"

Constraint programming (CP)

Characterization of constraint programming (CP)

CP is a computational system based on constraints

How does it work?

- **Phase 1:**
  - Express the CO problem in terms of a discrete problem (variables + domains)
  - Define constraints among the variables ("post constraints")
  - The constraint solver tries to reduce the variable domains (*propagation*)
    - ... before solution construction
    - ... after solution construction
- **Phase 2:** Labeling
  - Search through the remaining search tree
  - Possibly add (post) additional constraints

CP: domain reduction (example)

Simple example

minimize $f(X, Y, Z) \mapsto \mathbb{R}$

suject to

$$X \in \{1, \ldots, 8\}$$
$$Y, Z \in \{1, \ldots, 10\}$$
$$X \neq 7,\ Z \neq 2$$
$$X - Z = 3Y$$

Constraint propagation (step 1)

- Use $X \neq 7$ and $Z \neq 2$
  1. $X \in \{1, \ldots, 6, 8\}$
  2. $Z \in \{1, 3, \ldots, 10\}$

CP: domain reduction (example)

Constraint propagation (step 2)

- Make use of $X - Z = 3Y$
  1. Because of the domains of X and Z: $X - Z < 8$
  2. $\Rightarrow$ $3Y < 8$
  3. $\Rightarrow$ $Y \leq 2$
  4. $\Rightarrow$ $Y \in \{1, 2\}$

Constraint propatation (step 3)

- Use again $X - Z = 3Y$
  1. Because of the reduced domain of Y: $3Y \geq 3$
  2. $\Rightarrow$ $X - Z \geq 3$
  3. $\Rightarrow$ $X \in \{4, 5, 6, 8\}$ y $Z \in \{1, 3, 4, 5\}$

Hybrid metaheuristics: topics

- **Hybrid Metaheuristics (HMs)**
  - Definition
  - Clasification
- **Examples**
  - Metaheuristics with **other metaheuristics** (ILS)
  - Metaheuristics with **constraint programming** (ACO)
  - Metaheuristics with **tree search** (VNS)
  - Metaheuristics with **problem relaxation** (TS)
  - Metaheuristics with **dynamic programming** (EC)
- New algorithms/approaches from my group: CMSA y ACO_neg

Questions?

MHs: no new idea since 2005!

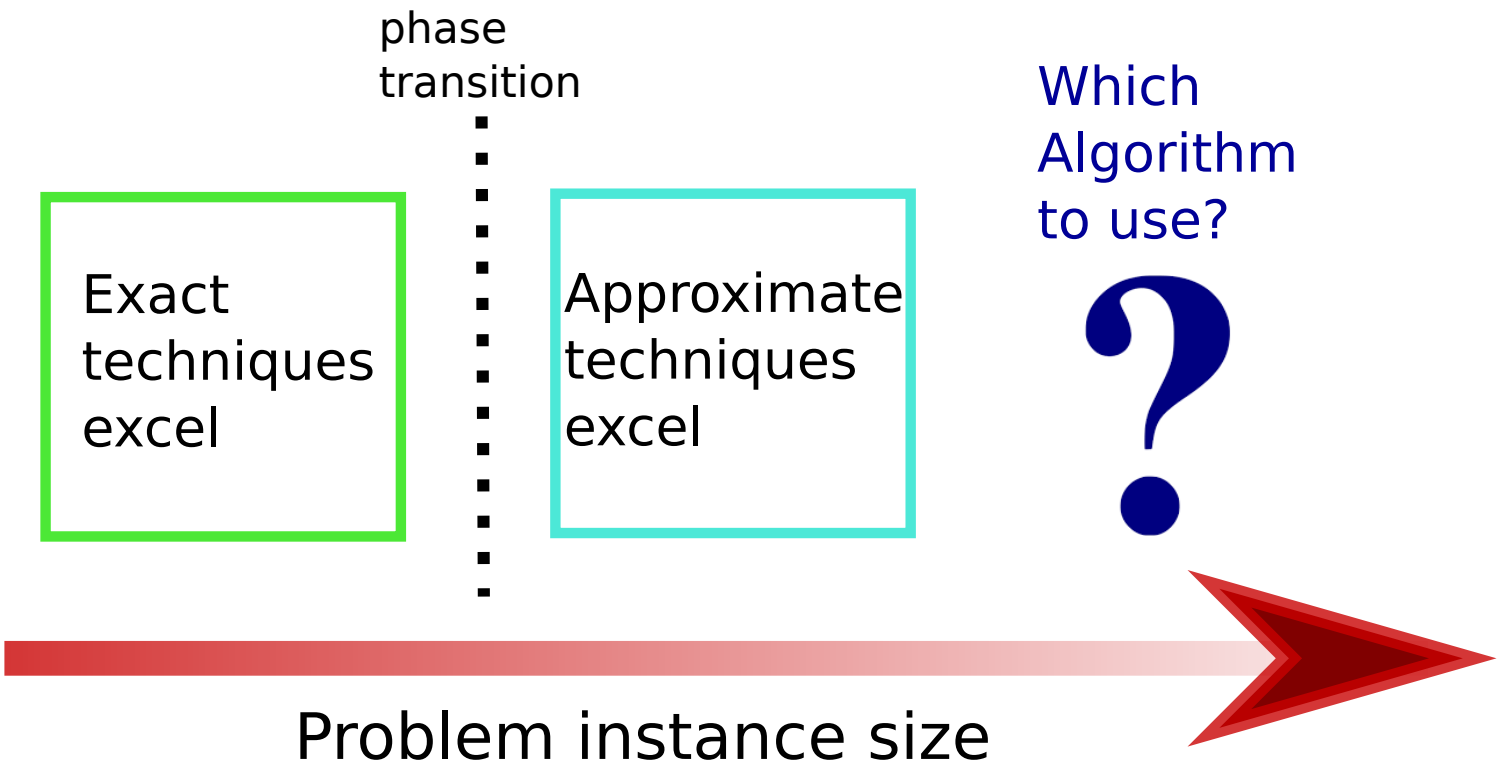
Important metaheuristics, year of introduction

- Simulated Annealing (SA) [Kirkpatrick, 1983]
- Tabu Search (TS) [Glover, 1986]
- Genetic and Evolutionary Computation (EC) [Goldberg, 1989]
- Ant Colony Optimization (ACO) [Dorigo, 1992]
- Greedy Rand. Adaptive Search Procedure (GRASP) [Resende, 1995]
- Particle Swarm Optimiation (PSO) [Eberhart, Kennedy, 1995]
- Guided Local Search (GLS) [Voudouris, 1997]
- Iterated Local Search (ILS) [Stützle, 1999]
- Variable Neighborhood Search (VNS) [Mladenović, 1999]
- Artificial Bee Colony (ABC) [Karaboga, 2005]

Note

No essentially new metaheuristics were introduced after 2005!!!

Hybrid metaheuristics: reasons for being



Note

Hybrid algorithms that take advantage of synergies between exact algorithms and approximate algorithms often excel for large-scale problems.

Hybrid MHs: characterization

What exactly is a hybrid metaheuristic?

Problem: this is not well-defined!

Possible characterization

- Hybrid MHs are techniques that result from the combination of a metaheuristic with other techniques for optimization
- Options for such other techniques:
  - Metaheuristics
  - Branch & bound
  - Dynamic programming
  - Integer linear programming (ILP)

Hybrid MHs: history

Pathway towards hybrid MHs

- For a long time the different optimization communities co-existed quite isolated
- Hybrid approaches were developed already early, but only sporadically
- Only since about 20 years the published body of research grows significantly signficativa:
  1. **1999:** first celebration of the **CP-AI-OR** conference
  2. **2004:** first celebration of the **Hybrid Metaheuristics (HM)** workshop
  3. **2006:** first celebration of the **Matheuristics Workshops**

Consequence

Nowadays the term hybrid metaheuristic identifies a new line of research.

Hybrid MHs: classification

Articles about the classification of hybrid MHs

- C. Cotta. **A study of hybridisation techniques and their application to the design of evolutionary algorithms**, *AI Communications*, 11(3-4):223–224, 1998
- E. Talbi. **A taxonomy of hybrid metheuristics**, *Journal of Heuristics*, 8(5):541–565, 2002
- G. Raidl. **A unified view on hybrid metaheuristics**, In: *Proceedings of HM 2006*, volume 4030, Springer LNCS, pages 1–112, 2006
- E.-G. Talbi, ed. **Hybrid metaheuristics**, Vol. 434 of *Studies in Computational Intelligence*, Springer, 2013
- C. Blum and G. Raidl. **Hybrid Metaheuristics – Powerful Tools for Optimization**, Springer, 2016

Different angles for the classification

Concerning the types of algorithms that are combined

- **Metaheuristics with other metaheuristics**. Examples:
  1. Use of neighborhood-based MHs within population-based MHs
  2. Multi-level frameworks
- **Metaheuristics with problem-specific techniques**. Examples:
  1. Continuous optimization: use of gradient-based methods
  2. Simulations for approximating the objective function
- **Metaheuristics with other AI/OR techniques**. Examples:
  1. Large neighborhood search (LNS)
  2. Combinations of metaheuristics with constraint programming
- **Metaheuristics with a human interactor**

Different angles for the classification

Depth/level of the hybridization

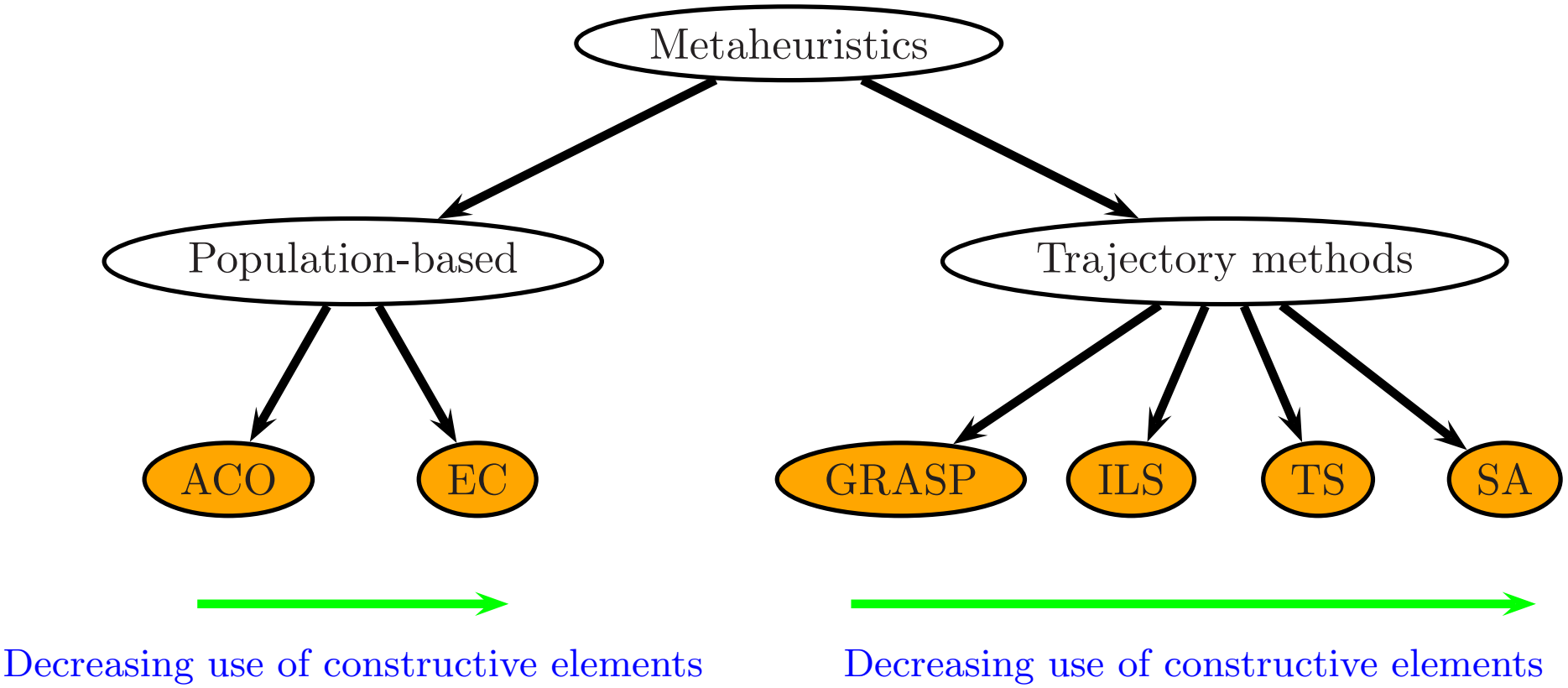
- **High-level:** weak coupling
  1. Algorithms retain their own identities
  2. No direct relationship of the internal workings of the algorithms
  3. Interaction over a well-defined interface
- **Low-level:** strong coupling
  1. Algorithms strongly depend on each other
  2. Individual components or functions are exchanged

Different angles for the classification

What is the control strategy?

- **Collaborative**
  1. **Homogeneous approaches:** several instances of the same algorithm
  2. **Heterogeneous approaches:** for example, A-Teams
- **Integrative**
  1. Solution merging (optimal crossover)
  2. Decoder-based approaches
  3. Large neighborhood search (LNS)
  4. Using metaheuristics for finding good upper bounds in branch & bound

Characteristics of different MHs



- **Advantage of population-based methods:** diversification
- **Advantage of trajectory methods:** intensification

Consequences for hybridization

Consequence

Most MH/MH hybrids incorporate trajectory methods into population-based techniques

Examples

- Application of local search to solutions constructed by ACO
- Application of local search to individuals in evolutionary algorithms (memetic algorithms)
- Population-based extension of iterated local search (ILS)
- Multi-level techniques

Iterated local search (ILS)

Pseudo code

$s_0 \leftarrow$ GenerateInitialSolution()
$s^* \leftarrow$ LocalSearch(s_0)
while termination conditions not met **do**
  $s' \leftarrow$ Perturbation(s^*)
  $s^{*\prime} \leftarrow$ LocalSearch(s')
  $s^* \leftarrow$ AcceptanceCriterion(s^*,$s^{*\prime}$)
end while
output: best solution found

Key components

- Perturbation mechanism
- Local search

Population-based ILS

Pseudo code

1: $P \leftarrow$ GenerateInitialPopulation(n)
2: Apply LocalSearch() to all $s \in P$
3: **while** termination conditions not met **do**
4: $P' \leftarrow P$
5: **for** all $s \in P$ **do**
6: $s' \leftarrow$ Perturbation(s)
7: $\hat{s}' \leftarrow$ LocalSearch(s')
8: $P' \leftarrow P' \cup \{\hat{s}'\}$
9: **end for**
10: $P \leftarrow$ Best n solutions from P'
11: **end while**

1

[1] T. Stützle. **Iterated local search for the quadratic assignment problem**, *European Journal of Operational Research*, 174(3):1529–1539, 2006

Multi-level techniques

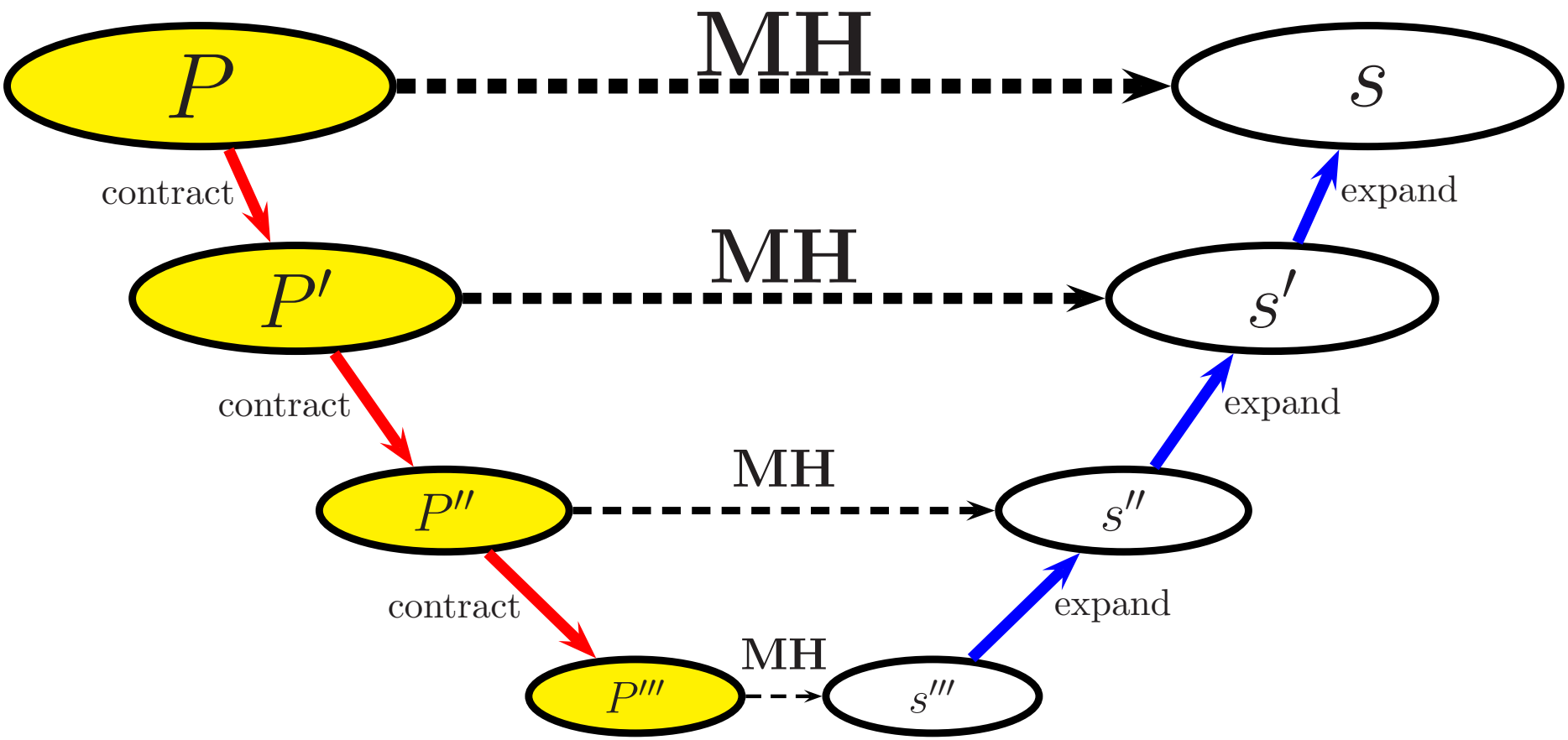
Articles

- C. Walshaw. **Multilevel refinement for combinatorial optimisation**, *Annals of Operations Research*, 131:325–372, 2004
- C. Walshaw. **Multilevel refinement for combinatorial optimisation: boosting metaheuristic performance**, In: *Hybrid Metaheuristics–An Emerging Approach to Optimization*, volume 114 of Studies in Computational Intelligence, pages 261–289, Springer Verlag, Berlin, Germany, 2008

General idea

- **First:** Iterative coarsening of the original problem instance
- **Then:** Find a solution to the coarsest level
- **Finally:** Iteratively refine this solution at each level

Multi-level technique



Multi-level techniques: TSP example

Example: on the board!

Application of the multi-level framework to the TSP. Here are some references to interesting applications.[abc]

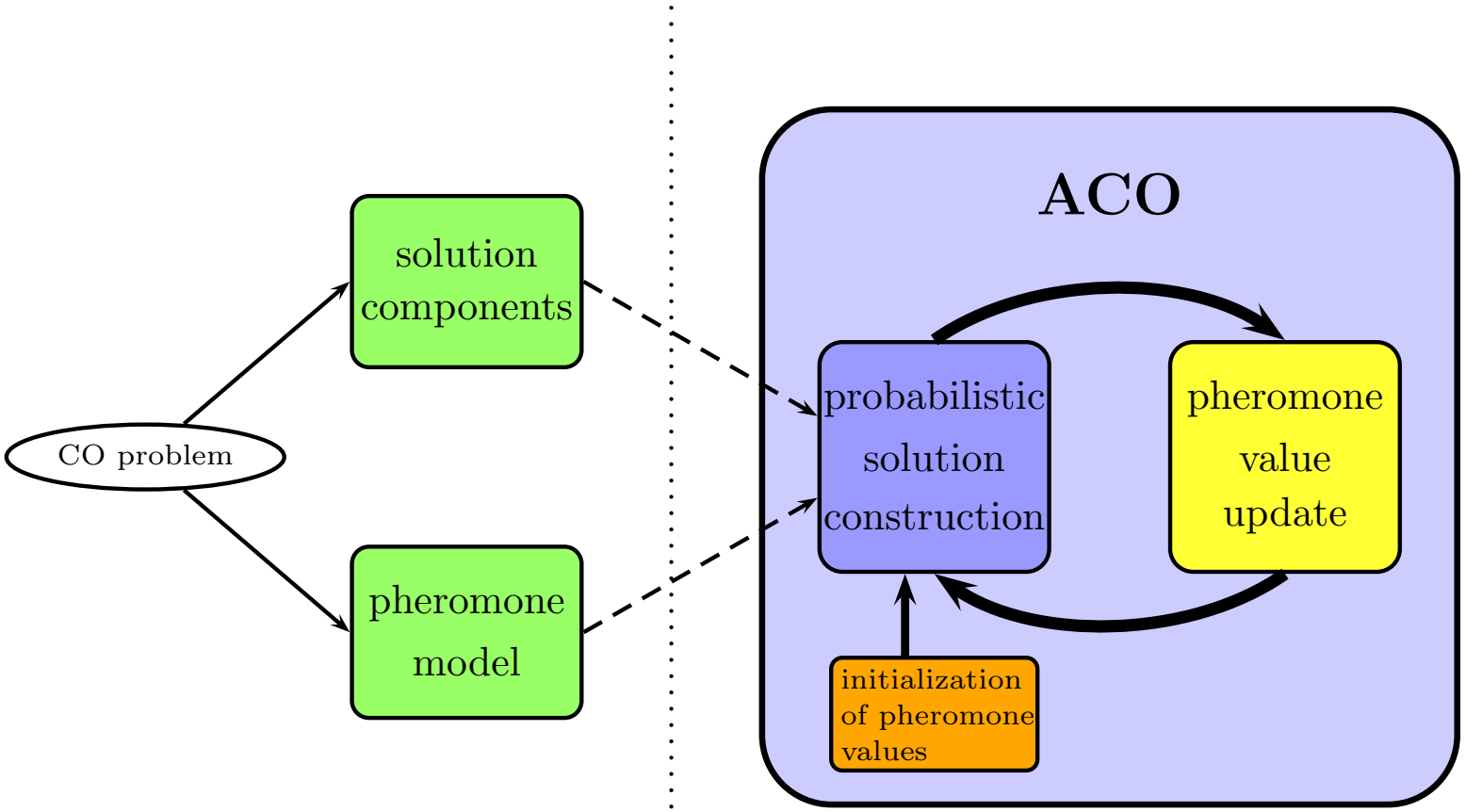
---

[a] C. Walshaw. **A multilevel approach to the travelling salesman problem**, *Operations Research*, 50(5):862–877, 2002

[b] T. G. Crainic, Y. Li, and M. Toulouse. **A first multilevel cooperative algorithm for capacitated multicommodity network design**, *Computers & Operations Research*, 33(9):2602–2622, 2006

[c] P. Lin, M. A. Contreras, R. Dai, and J. Zhang. **A multilevel ACO approach for solving forest transportation planning problems with environmental constraints**, *Swarm and Evolutionary Computation*, 28:78–87, 2016

Reminder: ACO

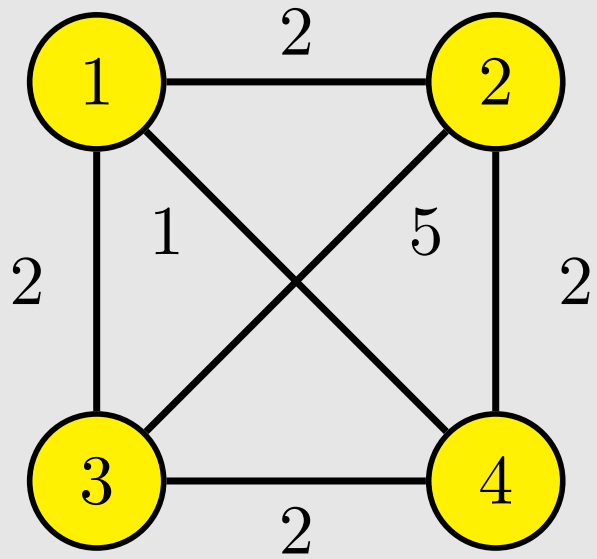


Note

As any other technique based on solution construction, ACO can be seen as a tree search algorithm

Example: ACO for the TSP

Example TSP instance

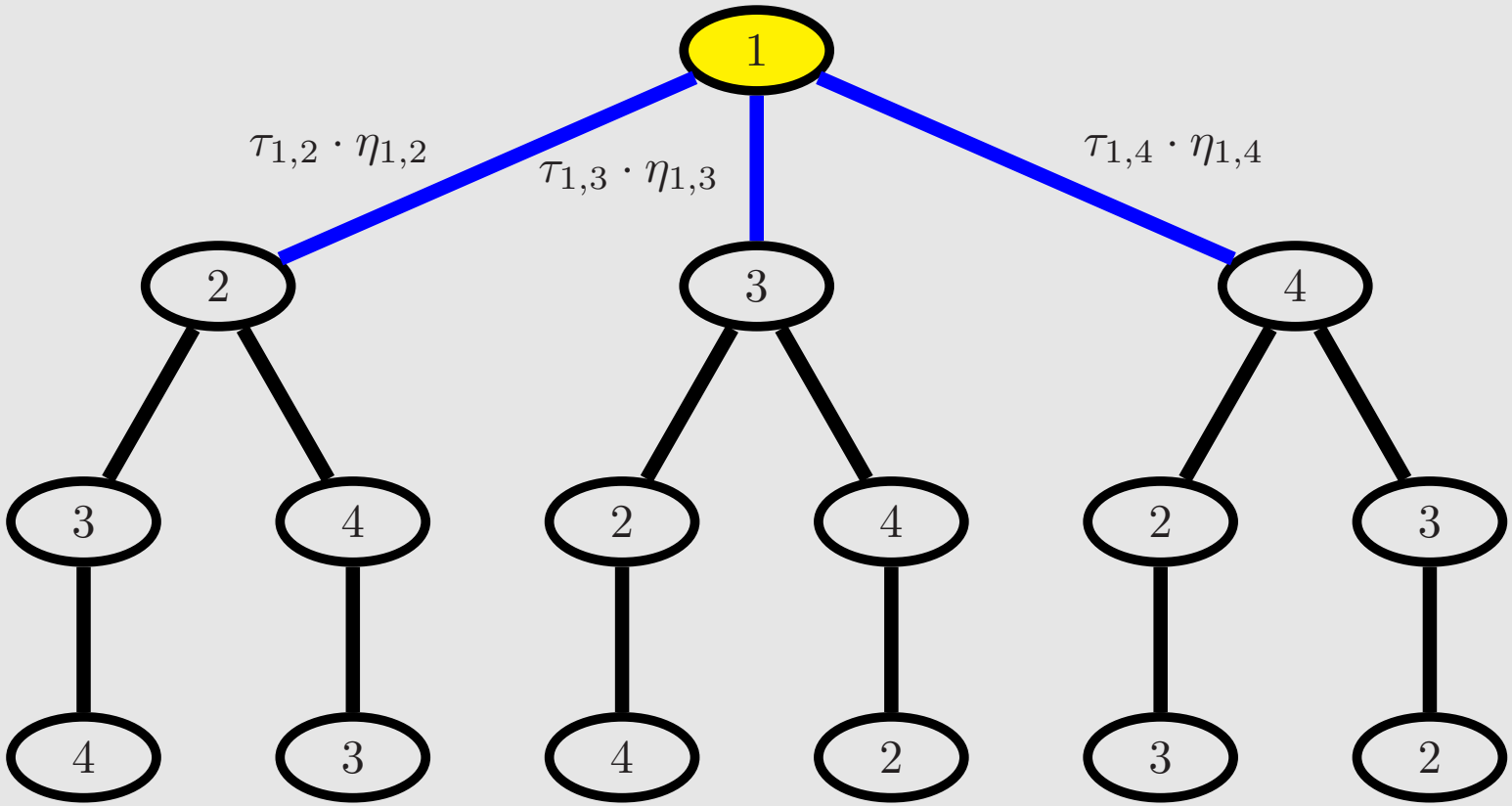


Solution construction mechanism

We will use the nearest-neighbor heuristics, starting every solution construction in city 1.

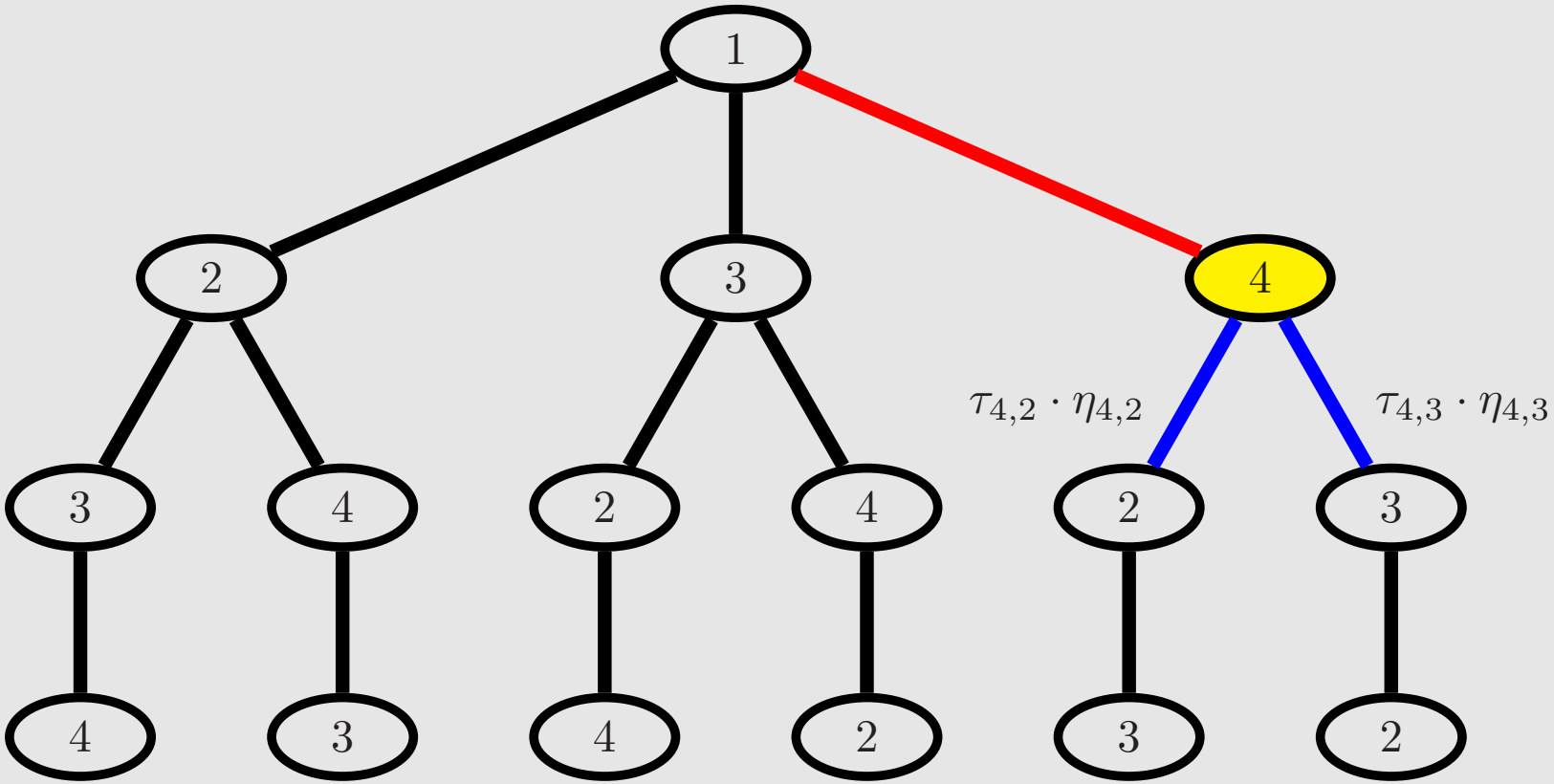
Example: ACO for the TSP

Construction step 1



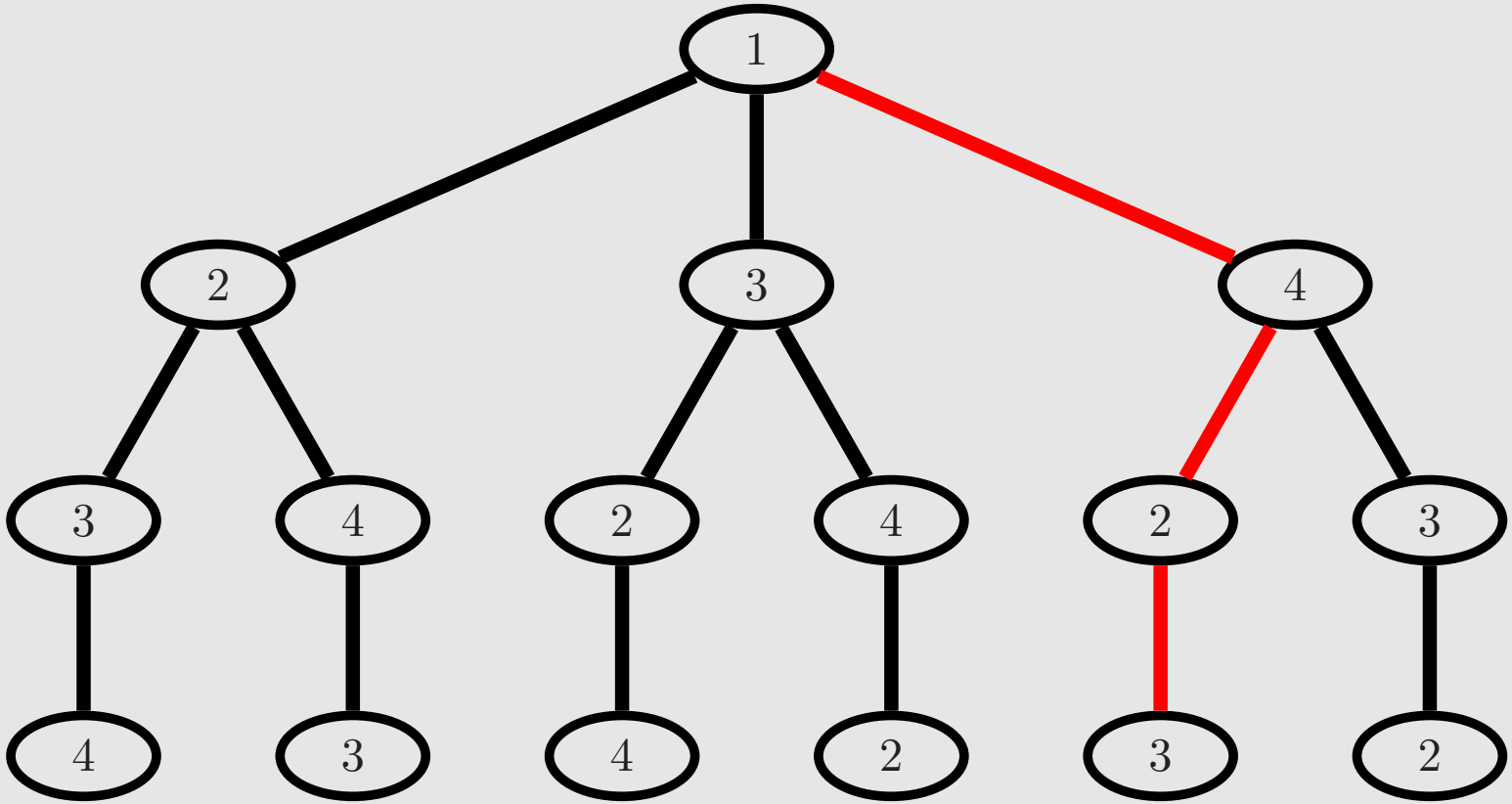
Example: ACO for the TSP

Construction step 2



Example: ACO for the TSP

Final solution constructed

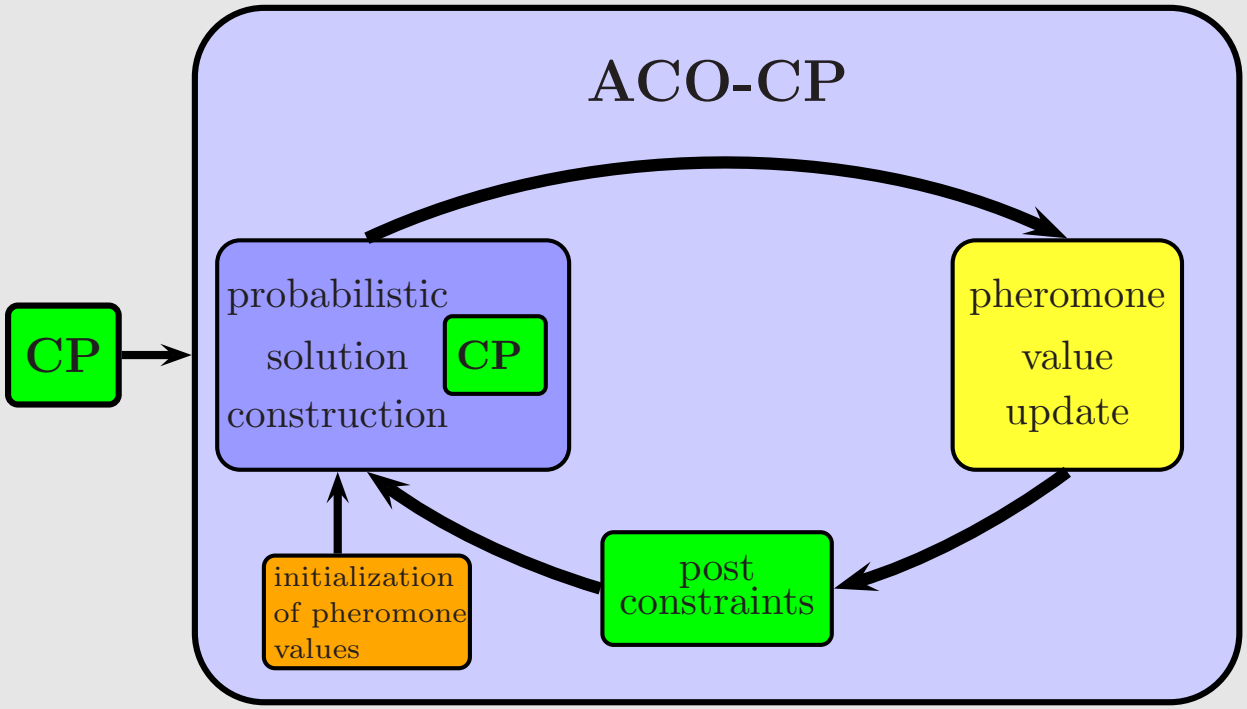


Hybrid between ACO and CP

General idea

- Successively reduce the variable domains by constraint propagation
- Let ACO search the reduced search tree

Graphical illustration



Hybrid between ACO and CP

Reasons for the benefits of such a hybrid

- **Advantage of ACO:** good in finding high quality solutions for moderately constrained problems
- **Advantage of CP:** good in finding feasible solutions for highly constrained problems
- **Advantage of ACO-CP:** good with intermediate number of feasible solutions factibles.[a]

[a] B. Meyer and A. Ernst. **Integrating ACO and Constraint Propagation**, In: *Proceedings of ANTS 2004*, volume 3172 of Springer LNCS, pages 166–177, 2004

Problems

- Constraint propagation takes a lot of time
- **Moreover:** constraint propagation is repeated many times

Hybrid techn.: large neighborhood search

Essential information

- **Large neighborhood search (LNS)** is known as one of the classical hybrid algorithms
- One of the first LNS applications deals with partitioning problems[a]
- A good recent survey can be found in[b]

---

[a] Ahuja, R. K., Orlin, J. B., & Sharma, D. Very largescale neighborhood search. International Transactions in Operational Research, 7(45), 301-317, 2000.

[b] David Pisinger, Stefan Ropke. Large Neighborhood Search, Handbook of Metaheuristics, International Series in Operations Research & Management Science Volume 272, 2019, pp 99-128

LNS: description (1)

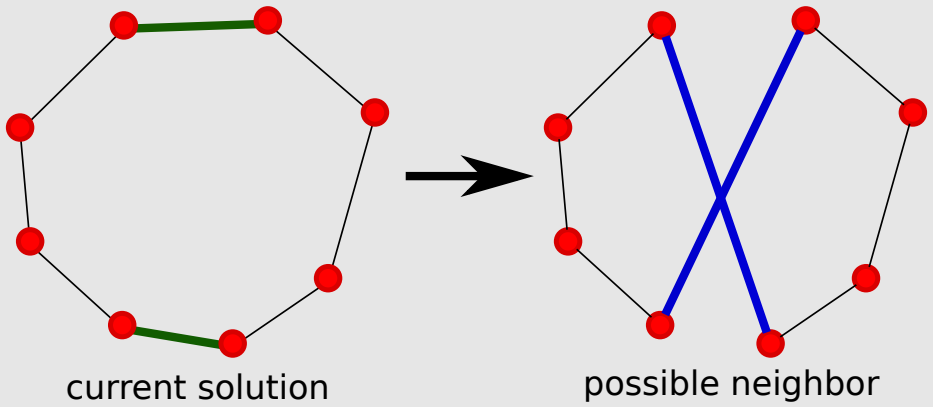
LNS is an algorithm based on local search

- **Crucial for any local search method:** Choice of a neighborhood
- **Usual in standard metaheuristics:** rather small neighborhoods, that is, each solution has a limited number of neighbors

Example of a small TSP neighborhood: 2-opt

Number of neighbors: $O(n^2)$

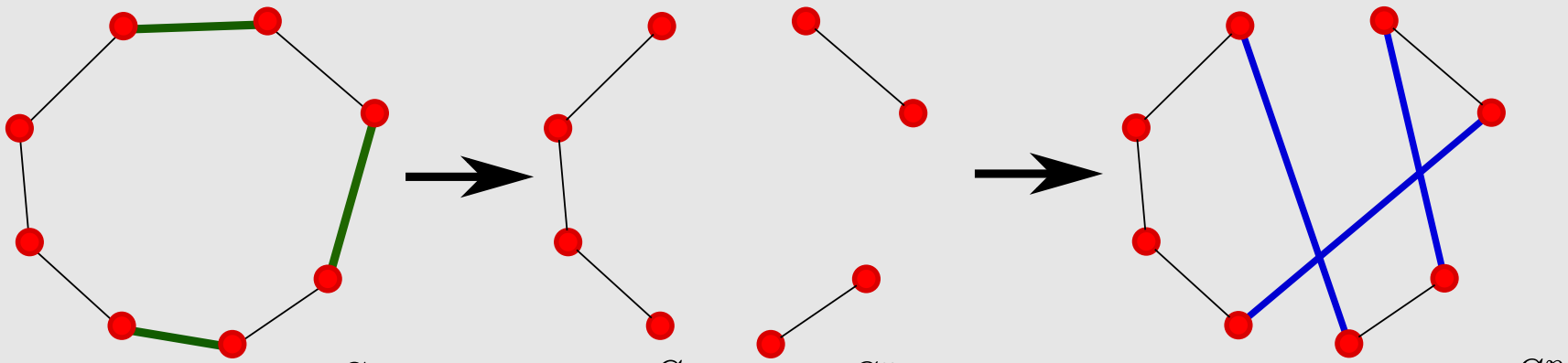
Traveling Salesman Problem: 2-opt move



LNS: description (2)

Example of a large neighborhood for the TSP

Traveling Salesman Problem: large neighborhood



LNS: description (3)

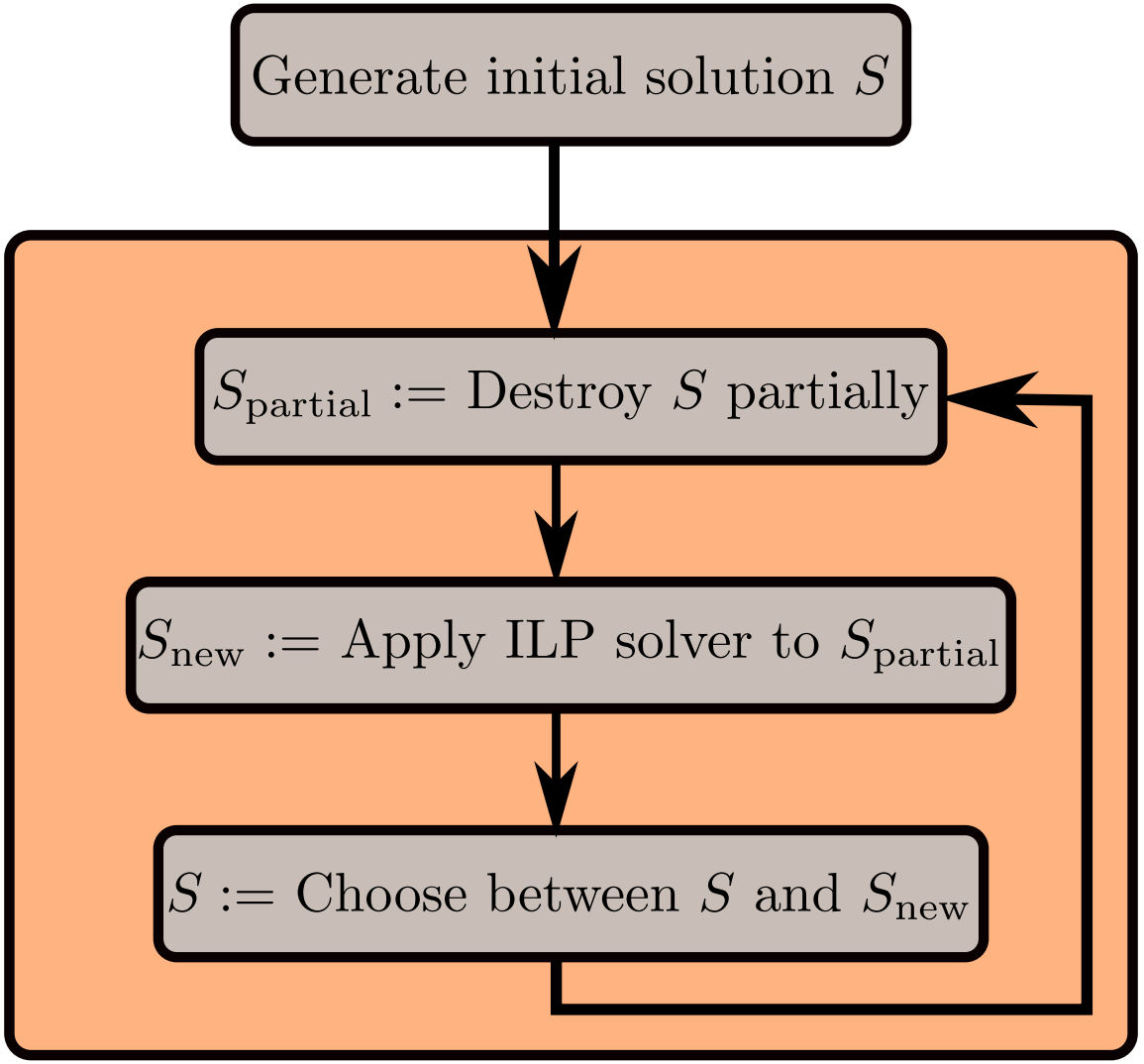
Trade-off in local search

- **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-scale 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 neighborhood

Ways to generate and explore large neighborhoods

- There are different ways to generate large neighborhoods. The most common way is to partially destroy an incumbent solution, with a subsequent reconstruction
- Large neighborhoods can be explored heuristically
- But also by making use of exact methods: for example, ILP solvers

Basic LNS based on solution destruction

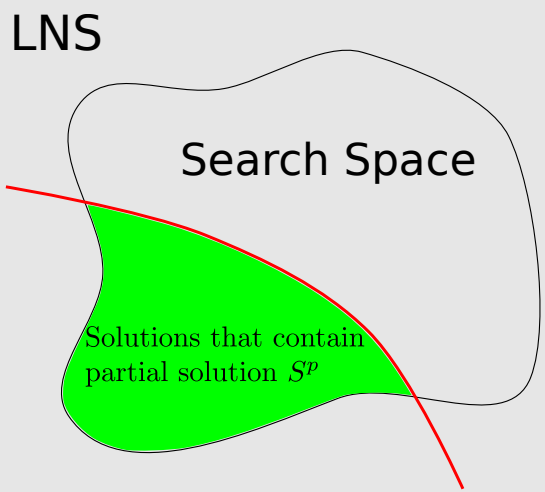


Crucial aspects of LNS

Coniderations

- **Important:** The application of an exact method in order to find the best solution containing a partial solution S_{parcial} means applying the exact method to a reduced search space.
- **Consequence:** In this way the exact method can be applied to larger problem instances.

Graphical illustration



LNS example: MWDS problem

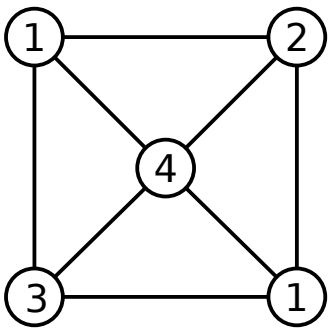
Minimum weight dominating set (MWDS) problem

- **Given:** an undirected graph $G = (V, E)$; each $v_i \in V$ has a weight $w(v_i) \geq 0$
- **Valid solutions:** Each $S \subseteq V$ is a valid solution, if and only if

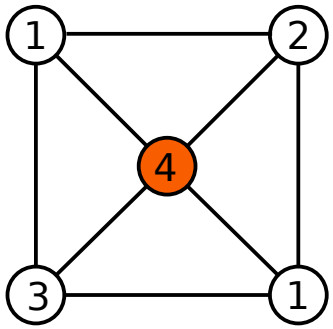
$$\forall\ v_i \in V\colon\ N[v_i] \cap S \neq \emptyset$$

- **Optimization objective:** finding a solution S^* that minimizes

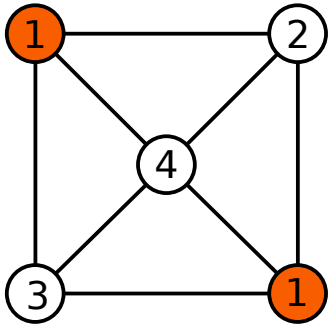
$$f(S^*) := \sum_{v_i \in S^*} w(v_i)$$



Given graph G



A valid solution



The optimal solution

LNS example: ILP is used as solver

ILP model

$$\begin{aligned} \text{min} \quad & \sum_{v_i \in V} x_i w(v_i) & \\ \text{subject to} \quad & x_i + \sum_{v_j \in N(v_i)} x_j \geq 1 & \forall\, v_i \in V \\ & x_i \in \{0,1\} & \forall\, v_l \in V \end{aligned}$$

Note

- In this ILP: the number of variables and constraints is linear in the problem parameters
- How to find the best solution that contains S_{parcial}?

  Add the following constraints to the ILP:

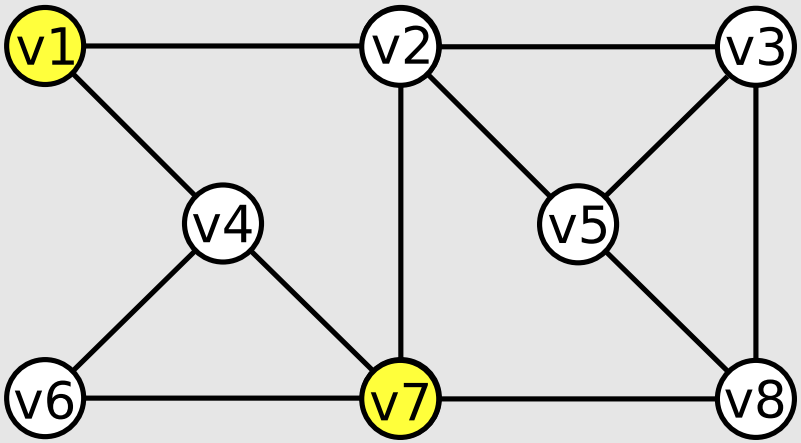
  $x_i = 1$ for all $v_i \in S_{\text{parcial}}$

LNS example: greedy heuristic

Definitions

- V_{cov}: the set of covered nodes (w.r.t. a partial solution))
- $d(v|V_{cov})$: current degree of node v (not considering already covered nodes)

Graphical illustration



$$S = \{v_1, v_7\}, \quad V_{cov} = \{v_1, v_2, v_4, v_6, v_7, v_8\}, \quad d(v_3|V_{cov}) = 1$$

LNS example: greedy heuristic

Pseudo code of GREEDY

1: **input:** undirected graph $G = (V, E)$ with node weights
2: $S := \emptyset$
3: $V_{\text{cov}} := \emptyset$
4: **while** $V_{\text{cov}} \neq V$ **do**
5: $\quad v^* := \text{argmax}_{v \in V \setminus V_{\text{cov}}} \left\{ \frac{d(v|V_{\text{cov}})}{w(v)} \right\}$
6: $\quad S := S \cup \{v^*\}$
7: $\quad V_{\text{cov}} := V_{\text{cov}} \cup N[v^*]$
8: **end while**
9: **output:** S

LNS example: partial sol. destruction

Partial destruction: standard method

Remove a certain percentage $perc_{\text{dest}}$ of the nodes of S

How to select the nodes to be removed?

- Destruction $type_{\text{dest}} = \text{random}$: random selection of nodes
- Destruction $type_{\text{dest}} = \text{biased}$: selection biased by the greedy function

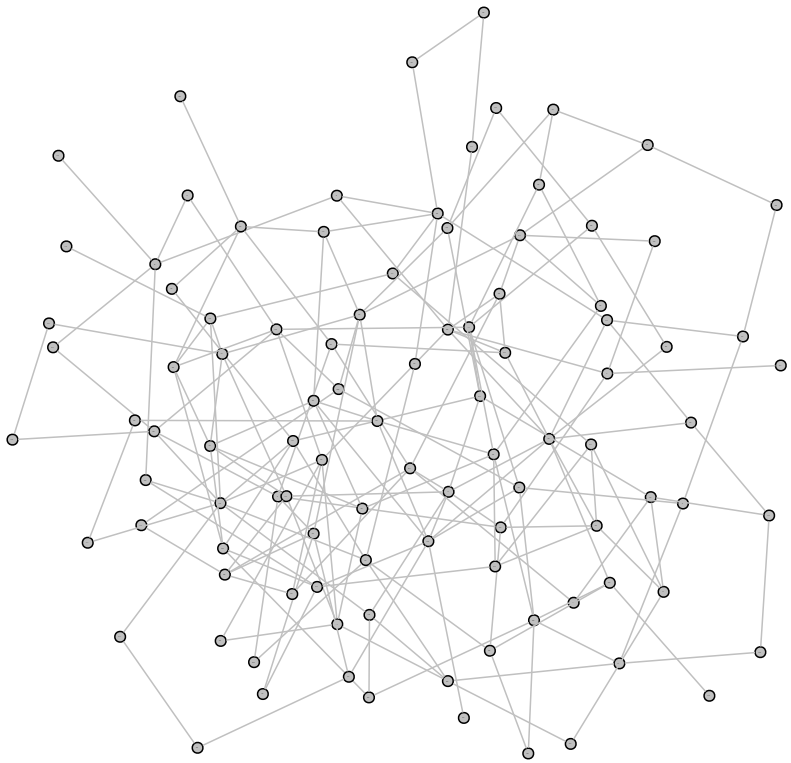
Choosing a value for $perc_{\text{dest}}$

- Dynamic scheme for choosing from $[perc^{l}_{\text{dest}}, perc^{u}_{\text{dest}}]$
- Initially $perc_{\text{dest}} := perc^{l}_{\text{dest}}$
- When no improving solution is found: $perc_{\text{dest}} := perc_{\text{dest}} + 5$
- When an improving solution is found or the upper bound is reached: $perc_{\text{dest}} := perc^{l}_{\text{dest}}$

LNS example: MWDS instances

Problem instances used for the experiments

- Random graphs with $|V| = \{100, 1000, 5000, 10000\}$ nodes
- Different edge probabilities e_p (low, medium, high):
  - For $|V| = 100$: $e_p \in \{0.03, 0.04, 0.05\}$
  - For $|V| > 100$: $e_p \in \{0.01, 0.03, 0.05\}$



LNS example: parameter tuning

Algorithm parameters and their domains

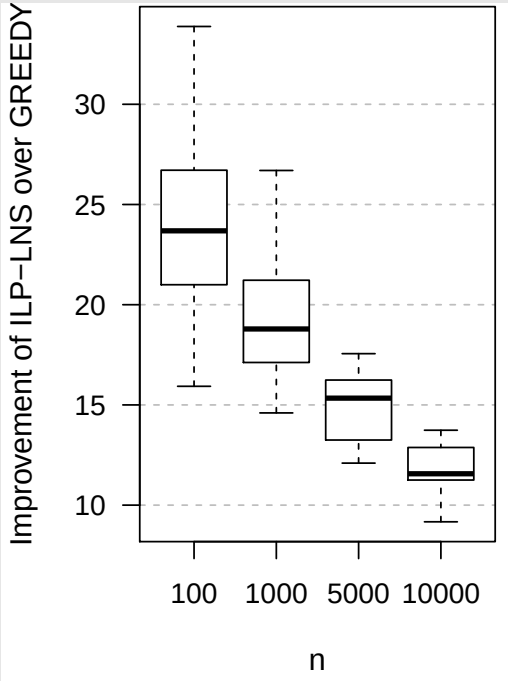
- $type_{dest}$ may be random or biased
- Upper and lower bounds $(perc^l_{dest}, perc^u_{dest})$ for the destruction percentage:
  1. (X, X) where $X \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$
  2. $(X, Y) \in \{(10, 30), (10, 50), (30, 50), (30, 70)\}$
- t_{max}: maximum CPU time for the ILP solver at each LNS iteration

Parameter values chosen by `irace`

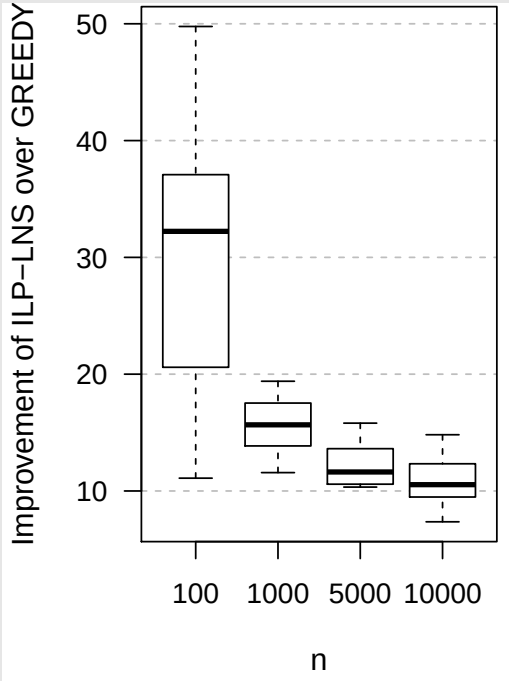
$\|V\|$	$type_{dest}$	$(perc^l_{dest}, perc^u_{dest})$	t_{max}
100	1	(60, 60)	2.0
1000	0	(90, 90)	10.0
5000	1	(50, 50)	5.0
10000	1	(40, 40)	10.0

LNS example: results

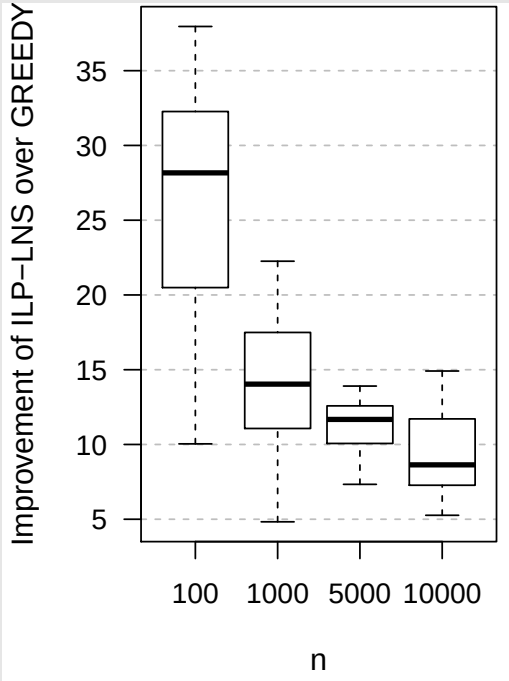
Improvement of LNS over GREEDY (in %)



low density



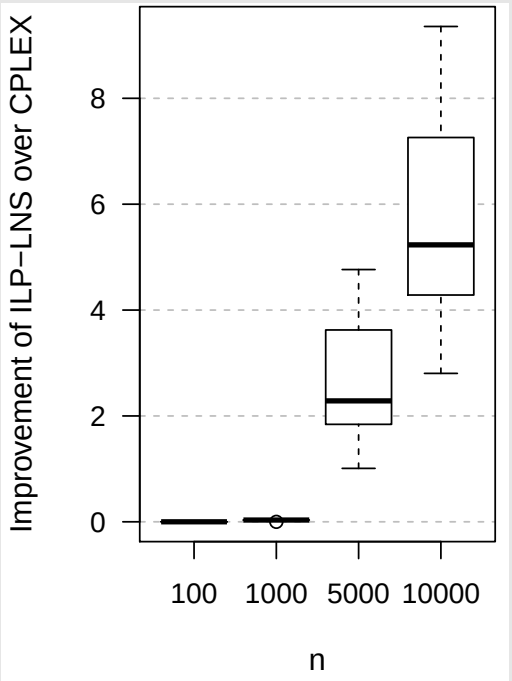
medium density



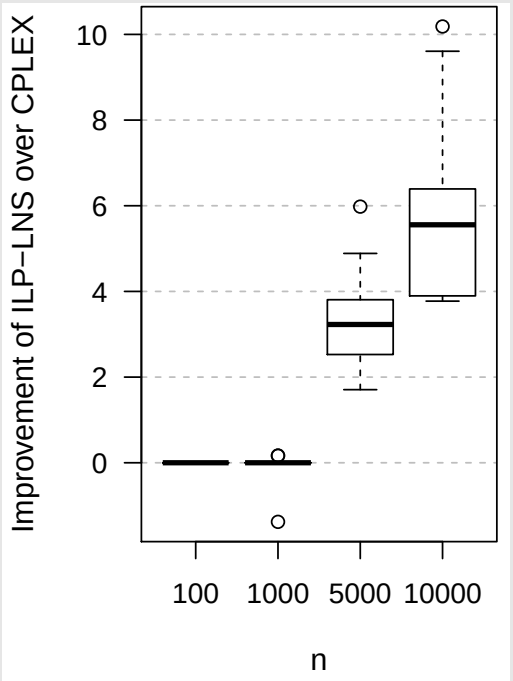
high density

LNS example: results

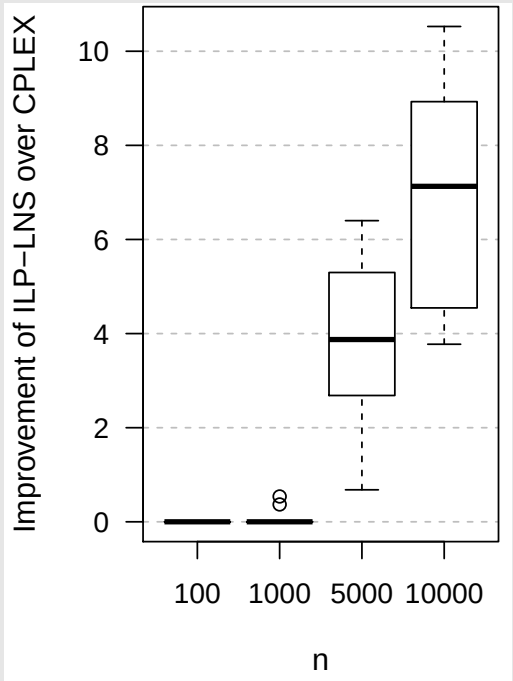
Improvement of LNS over CPLEX (in %)



low density



medium density



high density

2nd LNS example: optimization problem

Biological background

- **Given:** a set of haplotype sequences from a population of individuals
- **Goal:** study the evolutionary history of the chosen individuals
- **Important for** the discovery of the genetic basis of complex diseases

In case the population has evolved from a relatively small set of founders, the evolutionary history can be studied by trying to reconstruct the haplotype sequences from founder fragments

Problem

Generally, neither the founder sequences nor their number are known

2nd LNS example: optimization problem

The *Founder Sequence Reconstruction* (FSRP) problem

- **Given:** a set of m **recombinants** $\mathcal{C} = \{C_1, \ldots, C_m\}$, where C_i is a binary string of length n
- **Candidate solution:** a set of k **founders** $\mathcal{F} = \{F_1, \ldots, F_k\}$, where F_j is a binary string of length n
- $\mathcal{F}$ is a valid solution if $\mathcal{C}$ can be **reconstructed** from $\mathcal{F}$
- This is the case when each $C_i \in \mathcal{C}$ can be **decomposed** into a sequence of $p_i \leq n$ fragments $(Fr_{i1} Fr_{i2} \ldots Fr_{ip_i})$ such that each fragment Fr_{ij} appears at the same position in at least one of the founders
- Given $\mathcal{F}$, a **minimal decomposition**, which is characterized by a minimal number $f(\mathcal{F}) := \sum_{i=1}^{n} p_i - n$ of breakpoints, can be derived in polynomial time.
- **Optimization objective:** given k, find a valid solution $\mathcal{F}^*$ that minimizes $f(\cdot)$.

2nd LNS example: optimization problem

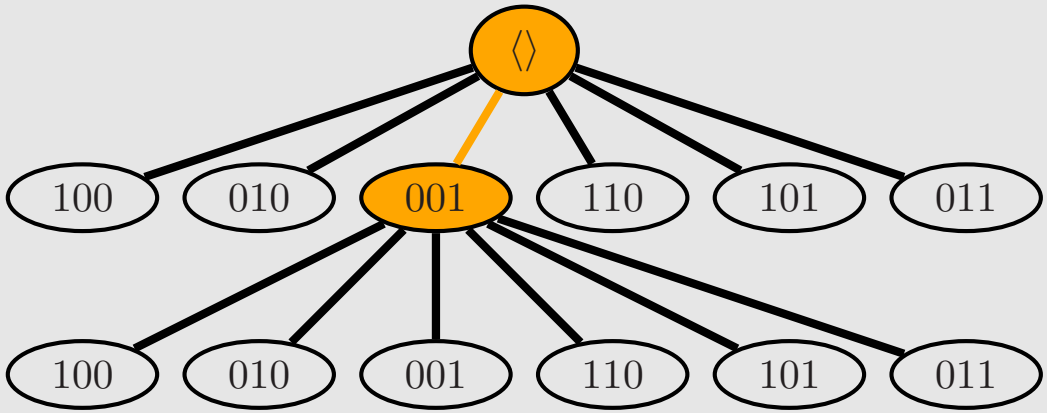
Small example instance

Recombinants $\mathcal{C}$	Founders $\mathcal{F}$	Reconstruction
1 1 0 1 1 0 1		b b b b b|c c
1 0 1 0 0 0 1	0 1 1 0 1 0 0	c c c c c c c
0 1 1 1 1 1 1	1 1 0 1 1 1 1	a a a|b b b b
0 1 1 0 1 0 0	1 0 1 0 0 0 1	a a a a a a a
1 1 0 0 0 1 1		b b b|c c|b b

2nd LNS example: exact technique

Specialized branch & bound for the FSRP: RECBLOCK

partial solution



search tree

2

[2] Wu, Y., Gusfield, D. **Improved algorithms for inferring the minimum mosaic of a set of recombinants**. In: *Proceedings of CPM 2007*, Volume 4580 of LNCS, Springer Verlag, Berlin (2007), pages 150–161

2nd LNS example: exact technique

Observation

Given some fixed founders, RECBLOCK can be used to obtain the optimal setting for the remaining founders

Example: 4 fixed founders $\{1, 2, 4, 7\}$, and 3 missing founders $\{3, 5, 6\}$

Reminder: VND

Observation

VND is a heuristic version of variable neighborhood search (VNS)

Pseudo code

input: a solution s, r_{max} neighborhoods
$r := 1$
while $r \leq r_{max}$ **do**
  $s' :=$ ChooseBestNeighbor$(s, \mathcal{N}_r)$
  if $f(s') < f(s)$ **then**
    $s := s'$, $r := 1$
  else
    $r := r + 1$
  end if
end while
output: an optimization solution s

2nd LNS example: the algorithm

LNS (based on VND) for the FRSP

1: **input:** a solution s, parameter $x < k$
2: $r := 1$
3: **while** $r \leq x$ **do**
4: $\hat{s} :=$ PartialDestruction(s, r)
5: $s' :=$ RECBLOCK$(\hat{s})$
6: **if** $f(s') < f(s)$ **then**
7: $s := s'$
8: $r := 1$
9: **else**
10: **if** termination conditions not met **then** $r := r + 1$
11: **end if**
12: **if** $r = x + 1$ **then** $r := 1$ **end if**
13: **end while**
14: **output:** an improved solution s

Example: hybridization based on relaxation

Note

Problem relaxations can be obtained (among others) by ...

- Simplifying the constraints of an ILP model
- Dropping some constraints from an ILP model
  Example: integrality constraints
- Removing some constraints and adding them as penalties to the objective function
  Example: *Lagrangian relaxation*

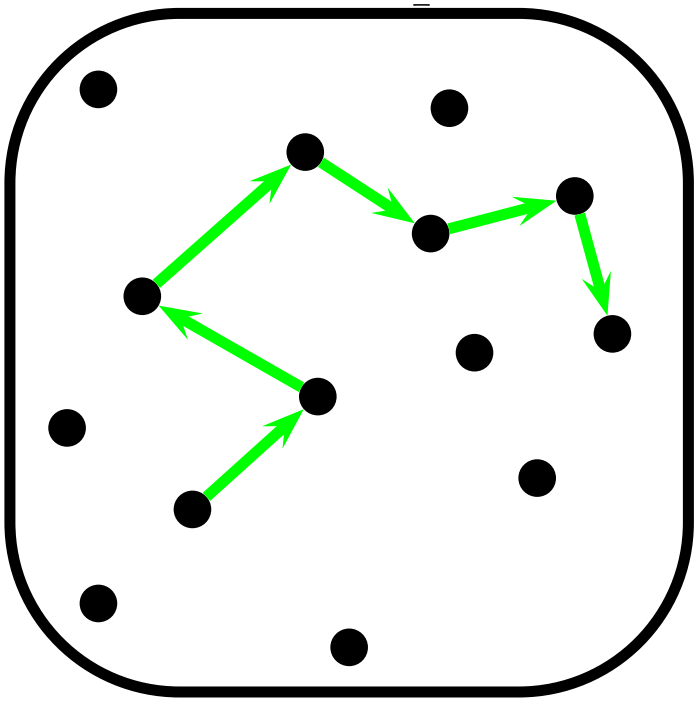
Use of relaxations

- As bounds for branch & bound algorithms
- As approximation for integer solutions
- As heuristic information for solution construction

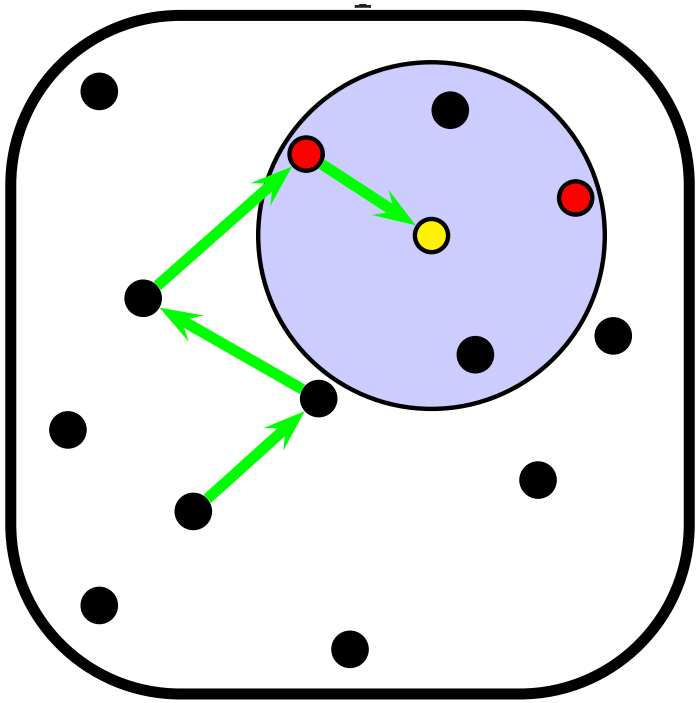
Reminder: tabu search (TS)

Main characteristics

- Use of tabu lists for storing solution features
- **Note:** tabu lists are used to avoid going back to already visited solutions



search space



selection of a neighbor

Example: hybrid TS

Specific Reference

M. Vasquez and Y. Vimont. **Improved results on the 0–1 multidimensional knapsack problem.** *European Journal of Operational Research*, 165(1):70–81, 2005

Characteristics of this hybrid method

- Collaborative hybridization approach
- The algorithm makes use of **two phases:**
  1. **Phase 1:** problem relaxation is used to produce a bunch of promising solutions
  2. **Phase 2:** tabu search is used to search around these solutions

Reminder: MDKP problem

Multidimensional Knapsack Problem (MDKP) problem

- **Given:**
  - n objects, each object i has a profit c_i
  - m resources, each resource j has a capacity b_j
  - Each object i has a requirement a_{ij} concerning resource j
- **Valid solutions:** every subset of the objects such that the capacity constraints of the resources are respected
- **Optimization objective:** maximizing the sum of the profits of the chosen objects

ILP model

$$\begin{aligned} & \textbf{max} \sum_{i=1}^{n} c_i \cdot x_i & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} \cdot x_i \leq b_j & j = 1, \ldots, m \\ & x_i \in \{0, 1\} & i = 1, \ldots, n \end{aligned}$$

Main ideas of phase 1 of hybrid TS

- Dropping the integrality constraints of the ILP model
- For all k such that $0 \leq k_{\min} \leq k \leq k_{\max} \leq n$ solve the following LP model:

$$\textbf{max} \sum_{i=1}^{n} c_i \cdot x_i$$

subject to:

$$\sum_{i=1}^{n} a_{ij} \cdot x_i \leq b_j \qquad j = 1, \ldots, m$$

$$0 \leq x_i \leq 1 \qquad i = 1, \ldots, n$$

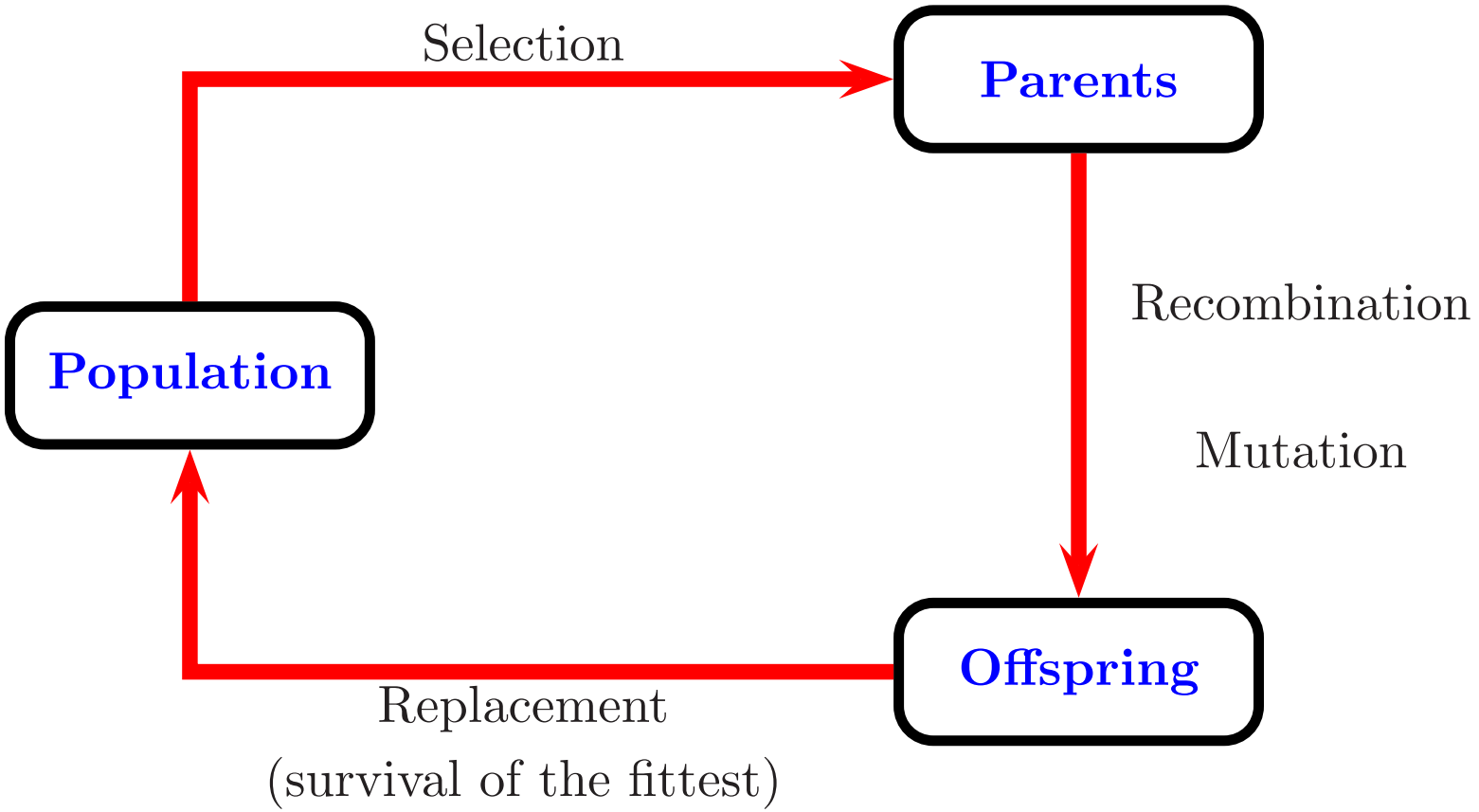
$$\sum_{i=1}^{n} x_i = k$$

Hybrid TS: phase 2

Main ideas of phase 2 of hybrid TS

- On the basis of the LP solutions from phase 1: produce integer solutions by rounding
- Use tabu search to search in the vicinity of these integer solutions
- Definition of vicinity: maximum allowed Hamming distance

Reminder: evolutionary algorithms (EAs)



KCT problem: a variation

Reference

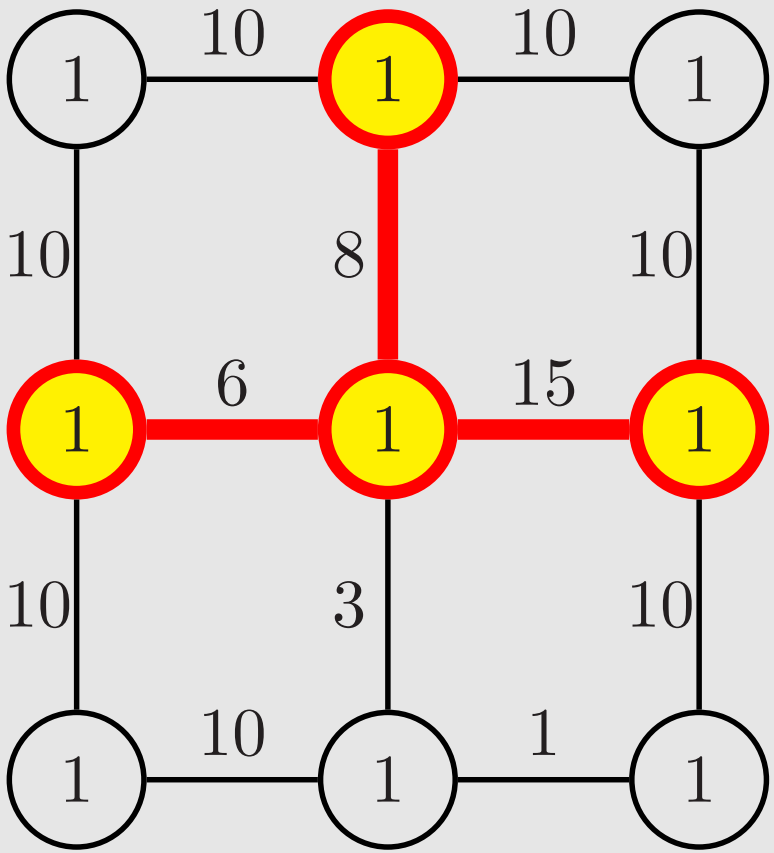
C. Blum. **A new hybrid evolutionary algorithm for the k-cardinality tree problem**, In: *Proceedings of GECCO 2006*, ACM Press, pages 515–522, 2006

KCT problem with node and edge weights

- **Given:** an undirected graph $G = (V, E)$,
- Edge weights w_e, $\forall\ e \in E$, and node weights w_v, $\forall\ v \in V$
- A constant $k < |V|$
- **Optimization objective:** find a tree T_k in G with exactly k edges that minimizes

$$f(T_k) = \left(\sum_{e \in E(T_k)} w_e \right) + \left(\sum_{v \in V(T_k)} w_v \right)$$

Graphical illustration of an instance and a solution

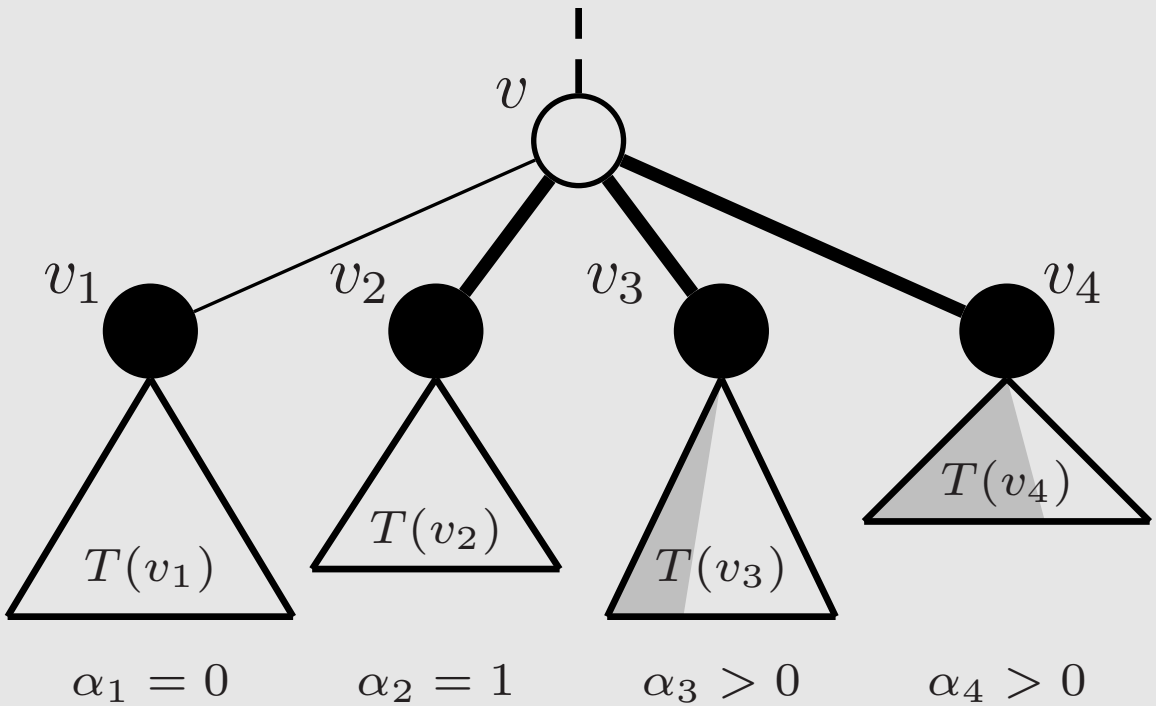


KCT problem variant: dynamic programming

Observation

There exists a dynamic programming algorithm for solving the KCT problem in case G itself is a tree. **Complexity:** $\mathcal{O}(k^2|V|)$

Graphical illustration



Making use of DP within an EA

Note

This dynamic programming algorithm is used in the recombination/crossover operator

Case 1: parent solutions T_1 and T_2 share at least one node

1. Merge T_1 and $T_2 \rightarrow$ a graph G_c
2. Generate a minimum spanning tree (MST) T of G_c
3. Utilize DP for deriving the best tree with k edges in T

Case 2: parent solutions T_1 and T_2 do not share any node

1. Extend T_1 step by step using a greedy algorithm until it contects with $T_2 \rightarrow$ a tree T that contains both T_1 and T_2
2. Utilize DP for deriving the best tree with k edges in T

Topics for the remainder of the course

Examples from our recent work

- **CMSA:** the algorithm *Construct, Merge, Solve & Adapt*
- **ACO_neg:** an ACO variant that makes use of negative learning
- **Beam-ACO:** an ACO variant that makes use of *beam search*

Motivation for the development of CMSA

In our experience ...

LNS works especially well when the following two conditions are fulfilled:

1. The number of solution components 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?

CMSA: main idea

Observation

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

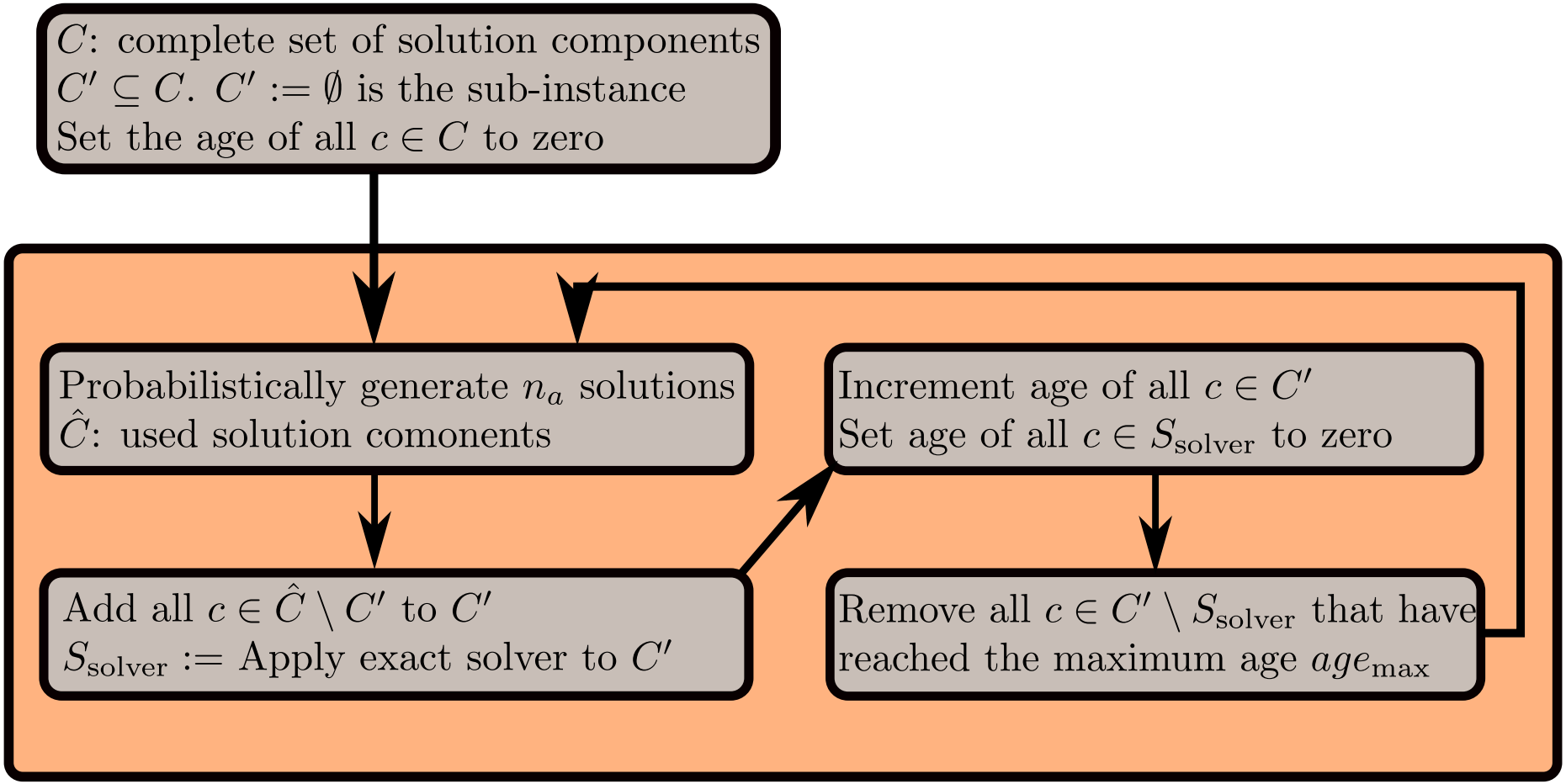
Idea

Exclude the presumably bad solution components from the ILP before solving it

Steps of one CMSA iteration

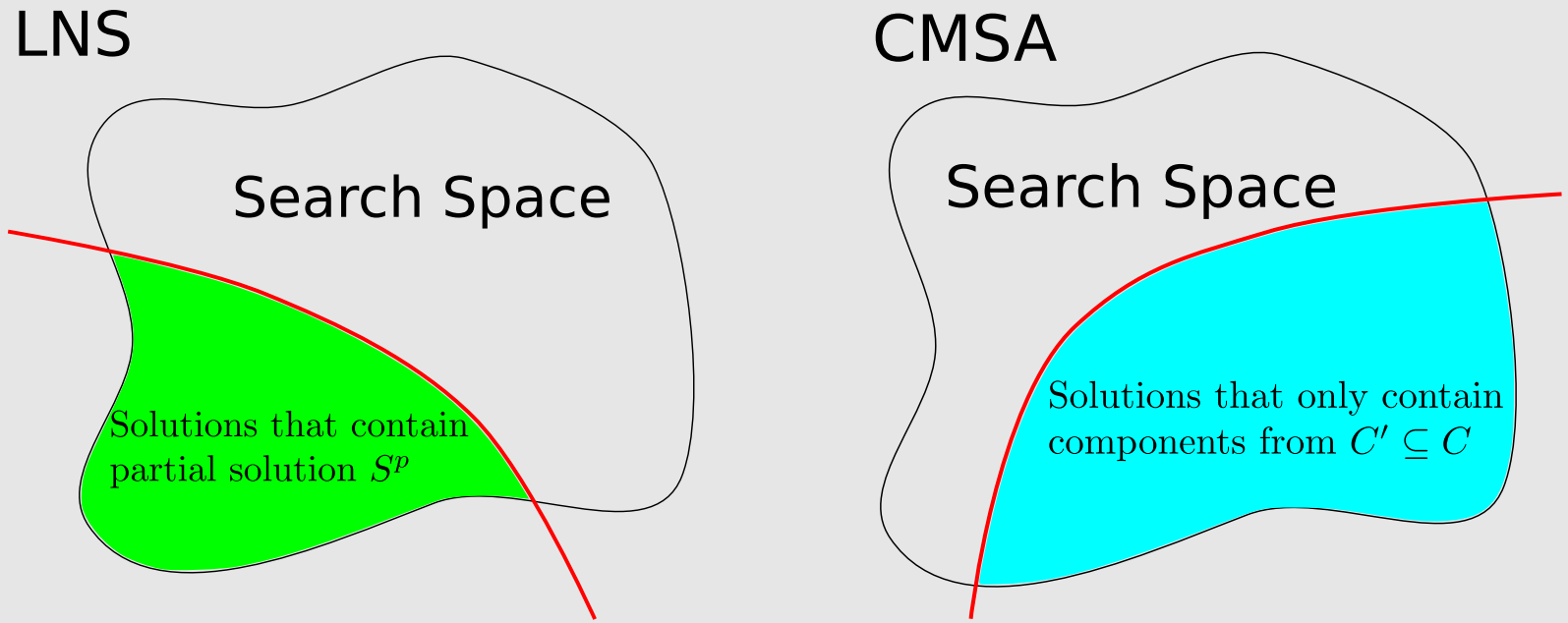
- Iteratively generate presumably good solutions in a probabilistic way
- Assemble a sub-instance from the used solution components
- Solve the sub-instance by means of an ILP solver
- Delete useless solution components from the sub-instance

CMSA: flow diagram



Main diff. between CMSA and LNS

How is the original problem instance reduced?



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

Application of CMSA: RFLCS problem

- The RFLCS problem is the LCS problem with an additional constraint: no letter may appear more than once in a valid solution
- **Proposed in:** 2010 in the journal *Discrete Applied Mathematics*
- **Complexity:** APX-hard (shown in the above article)
- **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

RFLCS problem: Best-Next heuristic

Best-Next: generates a solution from left to right

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

Question

How is Σ_t^{nd} defined?

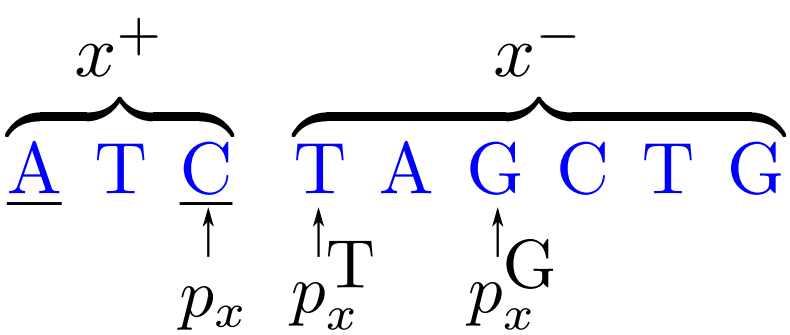
Best-Next: an example

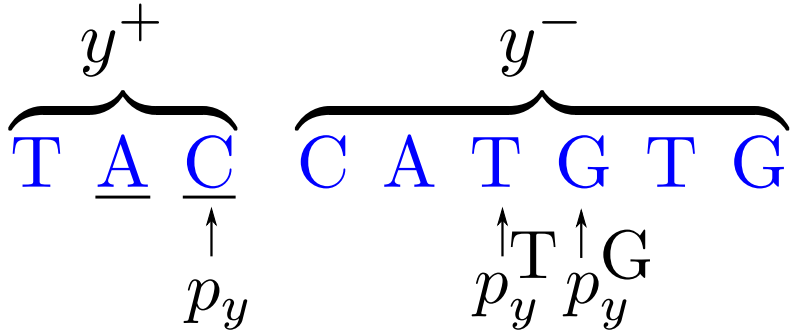
Example instance and partial solution

Given: $(x, y, \Sigma = \{A, C, T, G\})$ where

- $x = ATCTAGCTG$
- $y = TACCATGTG$

Partial solution: $t = AC$

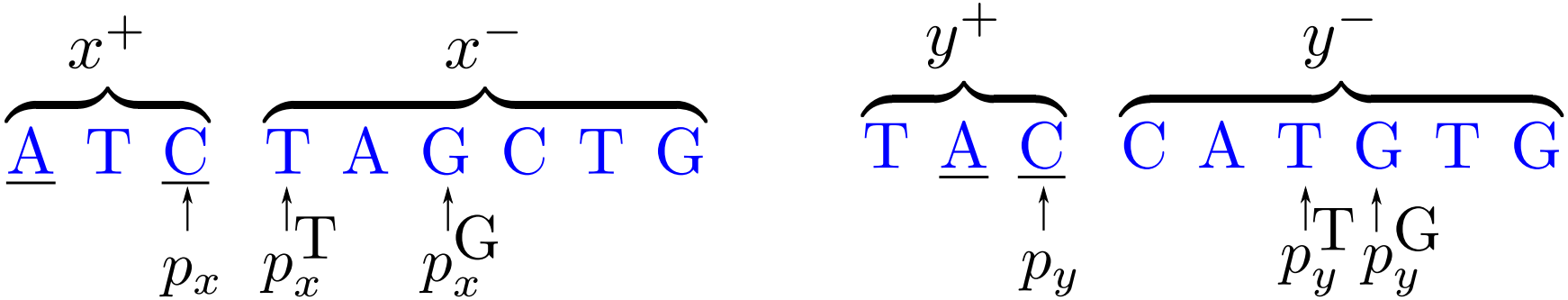




Therefore: $\Sigma_t^{nd} = \{T\}$

Best-Next: greedy function

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



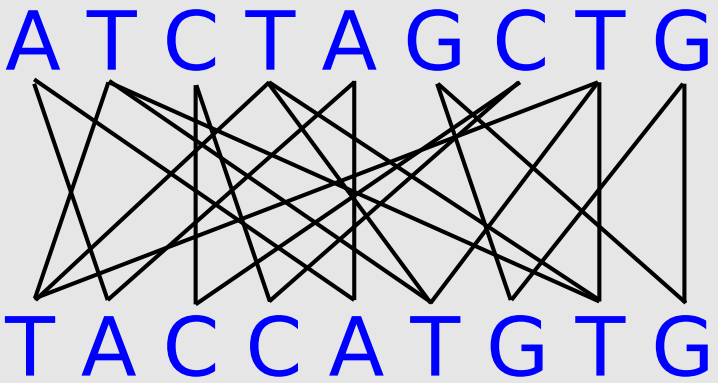
RFLCS problem: ILP model

RFLCS problem: variables of the ILP model

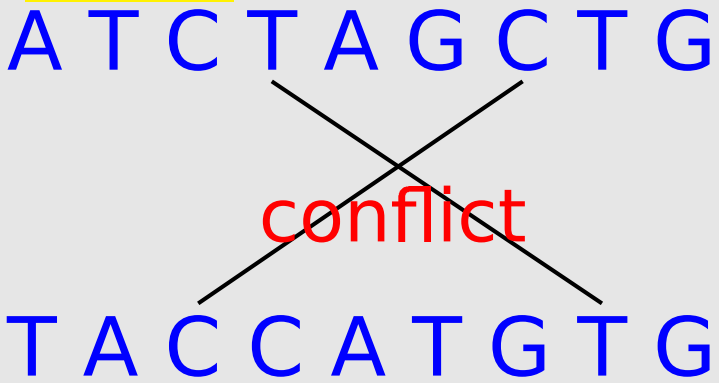
The ILP model consists of a binary variable $z_{i,j}$ for each pair of positions i in x and j in y such that $x[i] = y[j]$.

Example

the set of variables



conflicts between variables



RFLCS problem: ILP model

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

subject to:

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

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

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

Where ...

- $z_{i,j} \in Z_a$ if and only if $x[i] = y[j] = a$
- $z_{i,j}$ and $z_{k,l}$ are in conflict if and only if $(i < k$ and $j > l)$ or $(i > k$ and $j < l)$

RFLCS problem: experiments

Set 1 of problem instances

30 instances for each combination of ...

- Length of the input strings:
  $n \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$
- Alphabet size: $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$

Set 2 of problem instances

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: $rep \in \{3, 4, 5, 6, 7, 8\}$

Parameter tuning: CMSA

The parameters of CMSA are tuned by `irace` for each alphabet size.

RFLCS problem: CPLEX performance

Set 1

- Length of the input strings:
  $n \in \{32, 64, 128, 256, 512, 1024, 2028, 4048\}$
- Alphabet size: $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$

Set 2

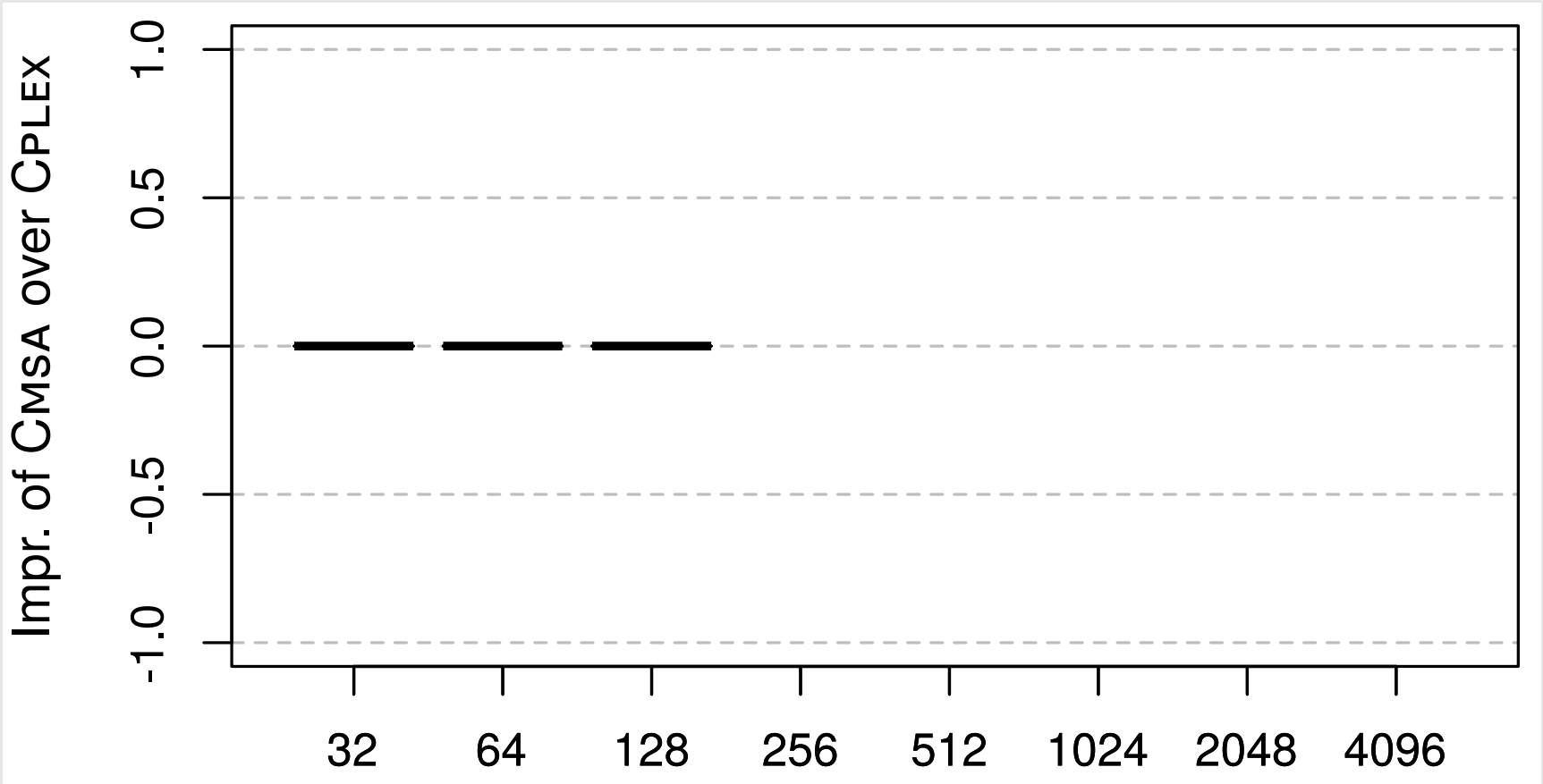
- Alphabet size: $|\Sigma| \in \{4, 8, 16, 32, 64, 128, 256, 512\}$
- Maximal number of repetitions of each letter: $rep \in \{3, 4, 5, 6, 7, 8\}$

Summary

CPLEX is able to solve nearly all existing problem instances from the literature to optimality

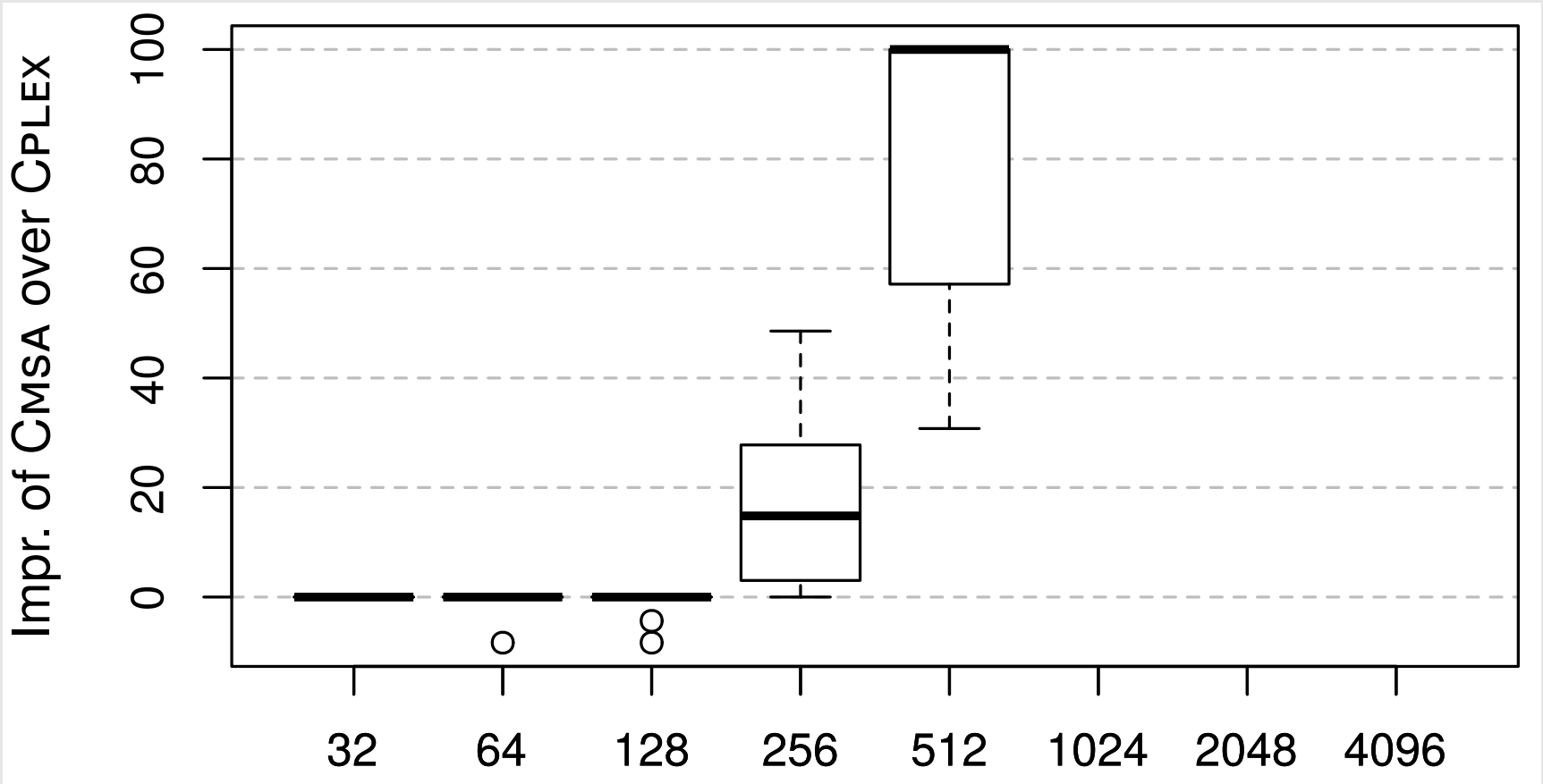
Performance of CMSA: set 1

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



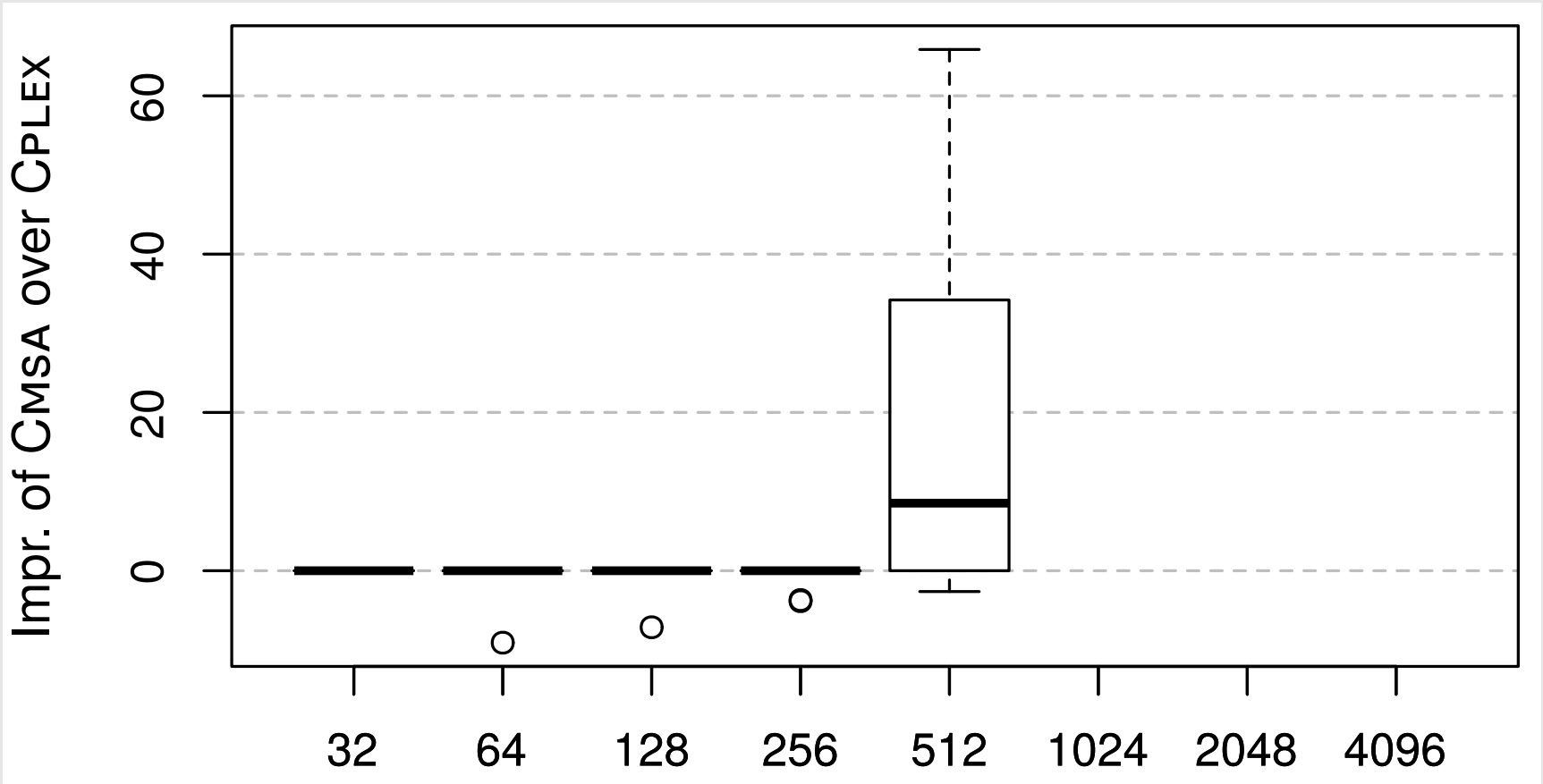
Performance of CMSA: set 1

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



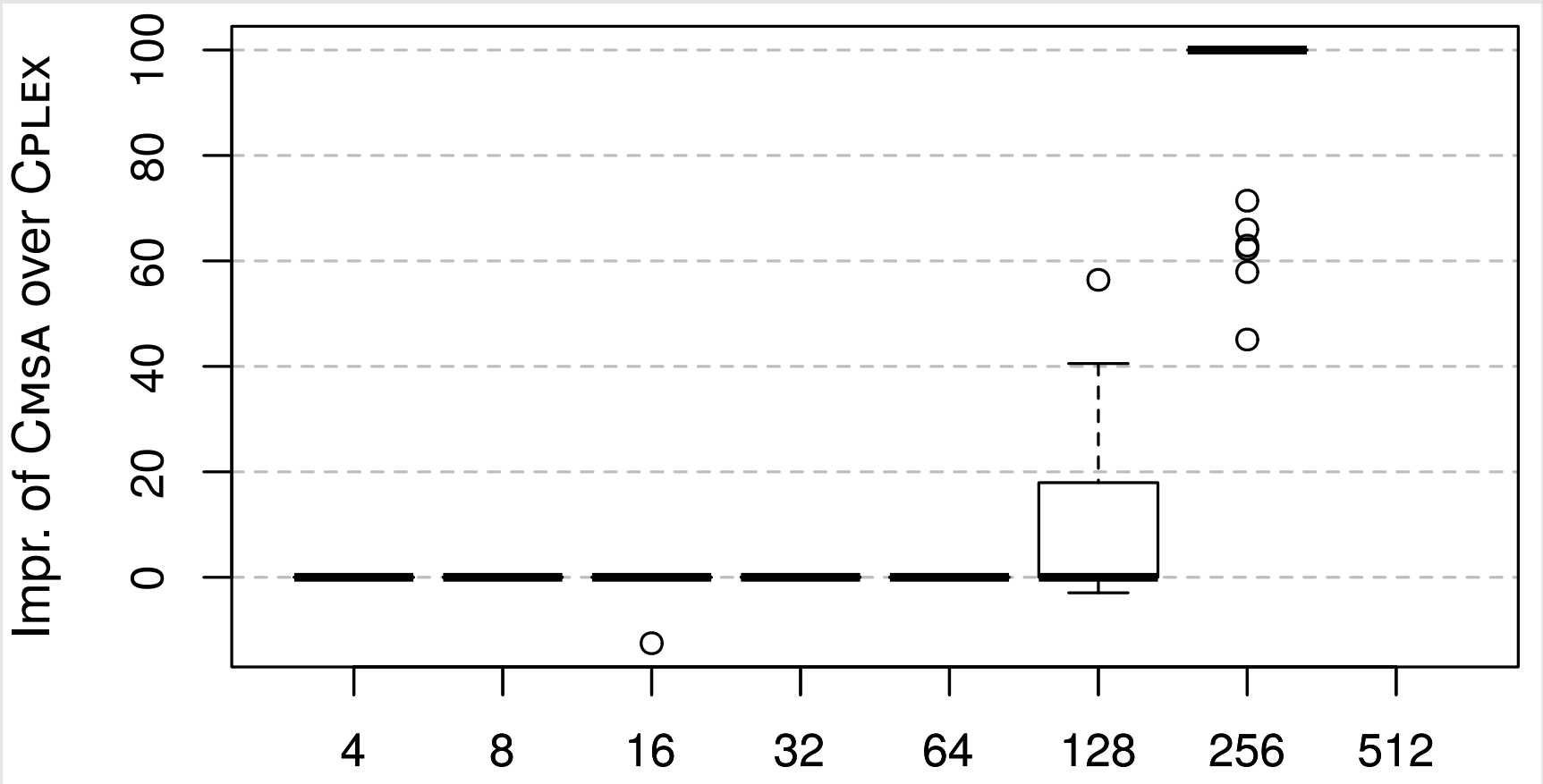
Performance of CMSA: set 1

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



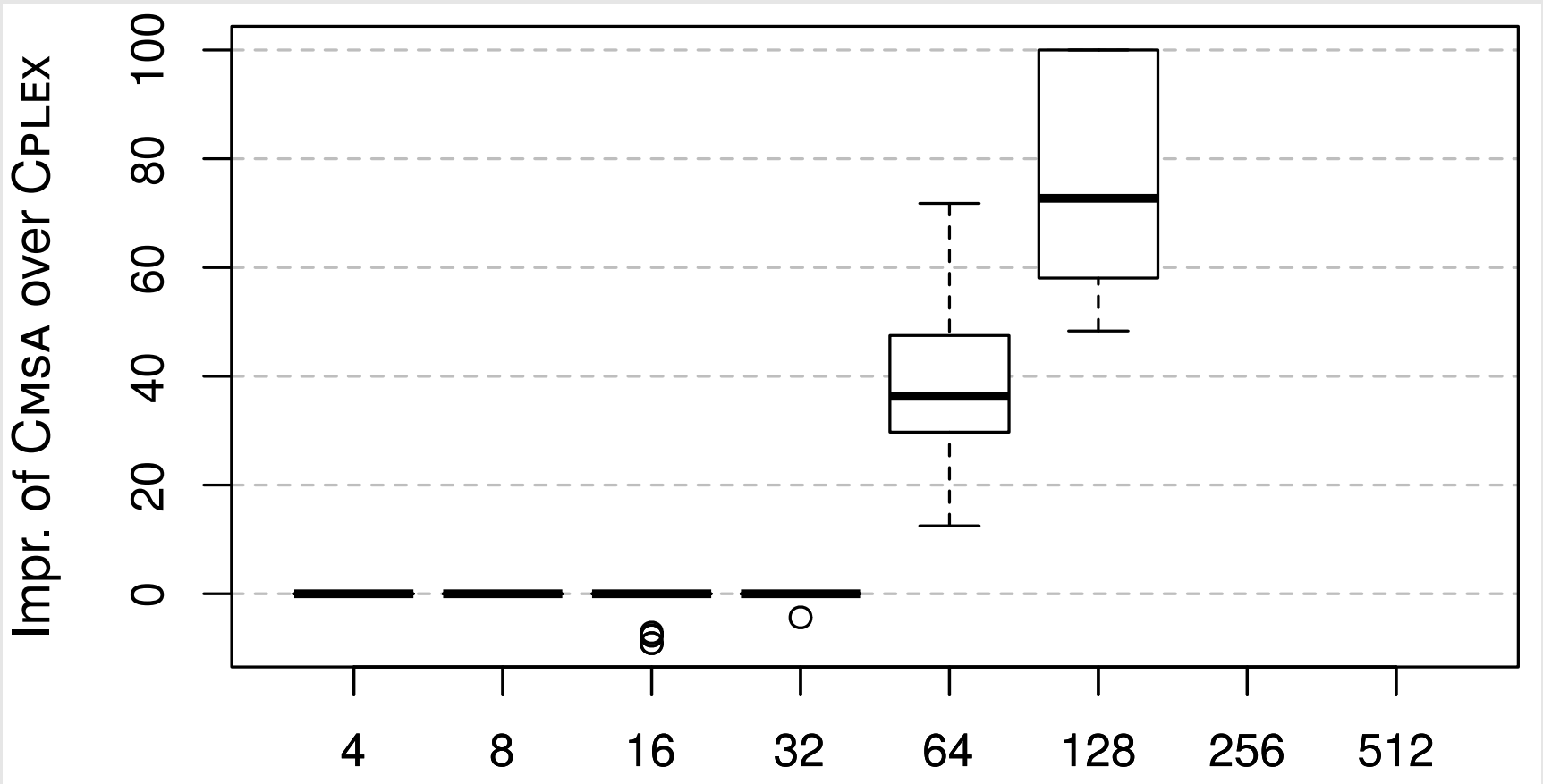
Performance of CMSA: set 2

Improvement of CMSA over CPLEX: maximally 3 occurrences



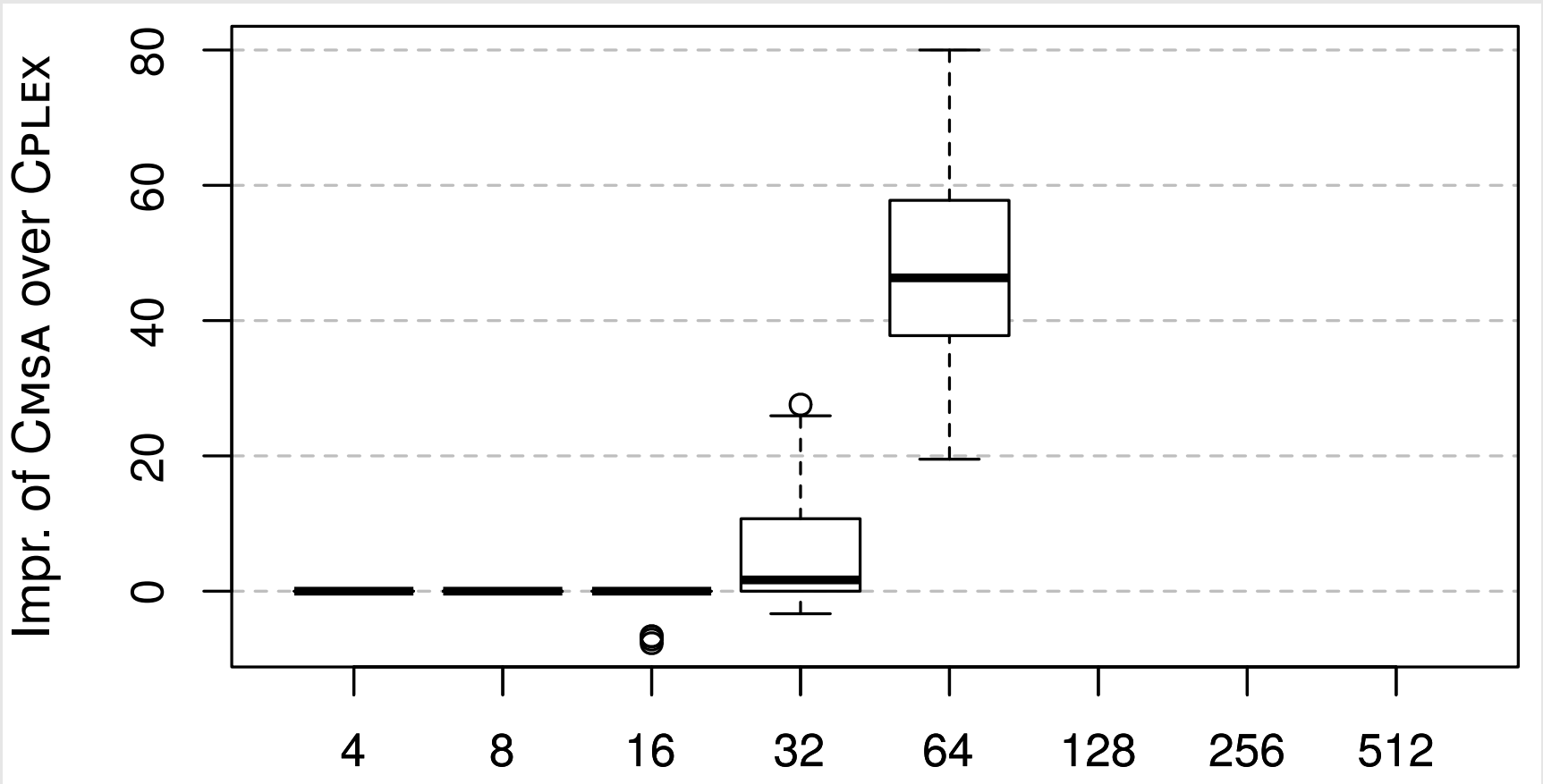
Performance of CMSA: set 2

Improvement of CMSA over CPLEX: maximally 6 occurrences



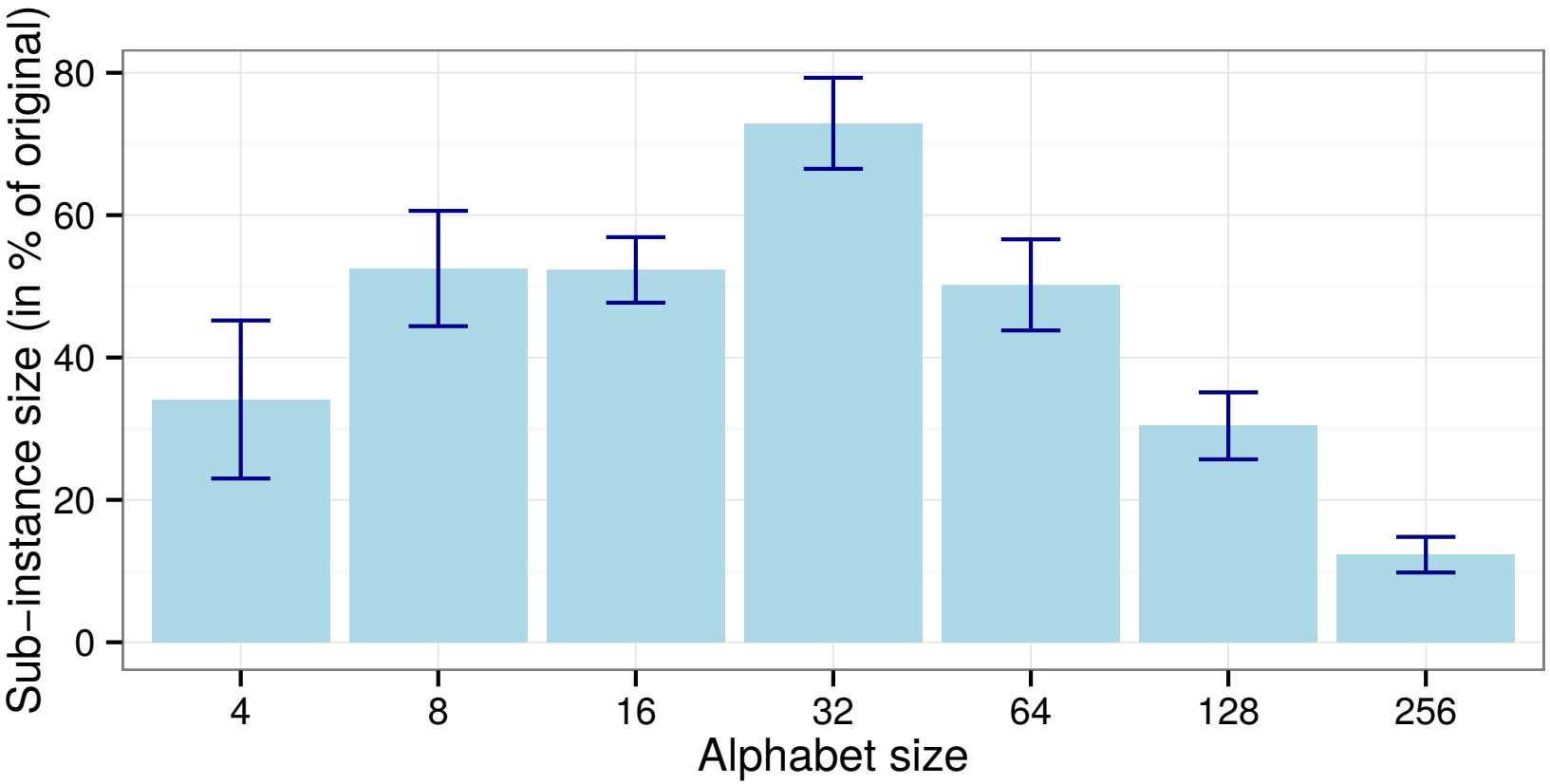
Performance of CMSA: set 2

Improvement of CMSA over CPLEX: maximally 8 occurrences



CMSA: size of sub-instances

Size of the sub-instances in CMSA



CMSA application 2: MCSP problem

Minimum common string partition (MCSP) problem

1. **Given:** Two **related sequences** of length n over alphabet Σ
2. **Note:** Sequences s_1 and s_2 are called related if and only if each letter appears the same number of times in each sequence

Valid solutions

- Generate a partition P_1 of sub-sequences of s_1 without overlap
- Generate a partition P_2 of sub-sequences of s_2 without overlap
- Solution $S = (P_1, P_2)$ is valid if and only if $P_1 = P_2$
- **Objective function:** $f(S) := |P_1| = |P_2|$
- **Optimization objective:** minimization

Additional information

- Introduced in 2005 in the context of genome rearrangement
- Complexity: NP-hard

MCSP problem: example

Example instance and solutions

- $s_1 := \text{AGACTG}$, $s_2 := \text{ACTAGG}$
- Trivial solution:
  - $P_1 = P_2 = \{\mathbf{A}, \mathbf{A}, \mathbf{C}, \mathbf{T}, \mathbf{G}, \mathbf{G}\}$
  - **Objective function value:** 6
- Optimal solution S^*:
  - $P_1 = P_2 = \{\mathbf{ACT}, \mathbf{AG}, \mathbf{G}\}$
  - **Objective function value:** 3

Algorithmic solutions from the literature

- 2005: a greedy heuristic
- 2007: an approximation algorithm
- 2008: a *fixed-parameter tractability (FPT)* study
- 2013: ant colony optimization metaheuristic

Model of the MCSP problem

Definition: given s_1 and s_2 ...

- A common block b_i is a triplet $(t_i, k1_i, k2_i)$ where t_i is a sequence ...
  1. starting at position $1 \leq k1_i \leq n$ in s_1
  2. starting at position $1 \leq k2_i \leq n$ in s_2
- B is the set of all common blocks of s_1 and s_2
- Any (partial) solution S is a subset of B such that
  1. $\sum_{b_i \in S} |t_i| = n$ (in case of a **complete solution**)
  2. $\sum_{b_i \in S} |t_i| < n$ (in case of a **partial solution**)
  3. For every pair $b_i, b_j \in S$: t_i and t_j do not overlap neither in s_1 nor in s_2

Example for this MCSP model

Input sequences

$s_1 =$ AGACTG and $s_2 =$ ACTAGG

Set B of all common blocks

$$\left\{\begin{array}{ll} b_1 = (\text{ACT}, 3, 1) & b_8 = (\text{A}, 3, 4) \\ b_2 = (\text{AG}, 1, 4) & b_9 = (\text{C}, 4, 2) \\ b_3 = (\text{AC}, 3, 1) & b_{10} = (\text{T}, 5, 3) \\ b_4 = (\text{CT}, 4, 2) & b_{11} = (\text{G}, 2, 5) \\ b_5 = (\text{A}, 1, 1) & b_{12} = (\text{G}, 2, 6) \\ b_6 = (\text{A}, 1, 4) & b_{13} = (\text{G}, 6, 5) \\ b_7 = (\text{A}, 3, 1) & b_{14} = (\text{G}, 6, 6) \end{array}\right\}$$

Solution and set of solution components

- Solution $\{\text{ACT}, \text{AG}, \text{G}\}$ corresponds to $\mathcal{S} = \{b_1, b_2, b_{14}\}$
- **Note:** The set B is the set of all solution components

MCSP problem: ILP model (1)

Input sequences

$s_1 =$ AGACTG and $s_2 =$ ACTAGG

$$B = \left\{ \begin{array}{l} b_1 = (\text{ACT}, 3, 1) \\ b_2 = (\text{AG}, 1, 4) \\ b_3 = (\text{AC}, 3, 1) \\ b_4 = (\text{CT}, 4, 2) \\ b_5 = (\text{A}, 1, 1) \\ b_6 = (\text{A}, 1, 4) \\ b_7 = (\text{A}, 3, 1) \\ b_8 = (\text{A}, 3, 4) \\ b_9 = (\text{C}, 4, 2) \\ b_{10} = (\text{T}, 5, 3) \\ b_{11} = (\text{G}, 2, 5) \\ b_{12} = (\text{G}, 2, 6) \\ b_{13} = (\text{G}, 6, 5) \\ b_{14} = (\text{G}, 6, 6) \end{array} \right\}$$

$$M1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

MCSP problem: ILP model (2)

ILP model: MCSP problem

$$\textbf{min} \sum_{i=1}^{m} x_i$$

subject to:

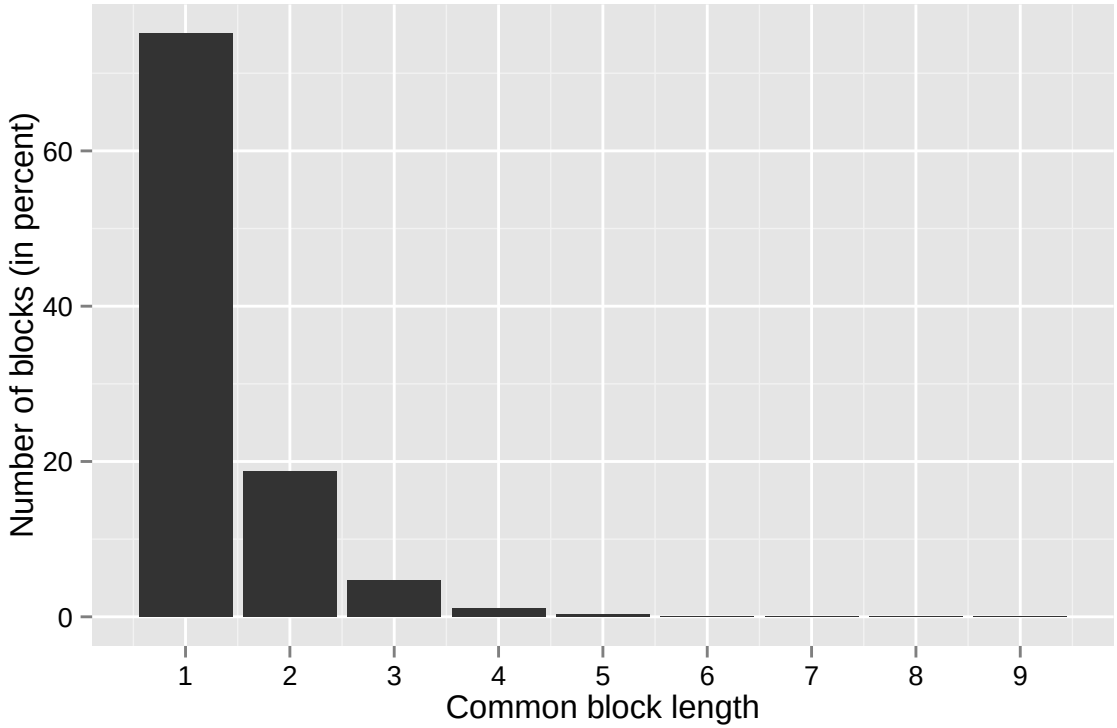
$$\sum_{i=1}^{m} |t_i| \cdot x_i = n$$

$$\sum_{i=1}^{m} M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n$$

$$\sum_{i=1}^{m} M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n$$

$$x_i \in \{0, 1\} \quad \text{for } i = 1, \ldots, m$$

MCSP problem: set of common blocks



Note

- An instance consists of many solution components
- The vast majority of common blocks of length 1 and 2 will not form part of good solutions

MCSP problem: greedy heuristic

Simple greedy heuristic

- At each construction step we add exactly one common block to the partical solution S_{parcial}
- Given a partial solution S_{parcial}, $N(S_{\text{parcial}}) \subset B$ denotes the set of common blocks that can be added to S_{parcial}

Pseudo code

1. $S_{\text{parcial}} := \emptyset$
2. **while** S_{parcial} is not a complete solution
   - Select a largest common block b_i from $N(S_{\text{parcial}})\}$
   - $S_{\text{parcial}} := S_{\text{parcial}} \cup \{b_i\}$

Note

This greedy heuristic is used in a probabilistic way within CMSA

Problem instances

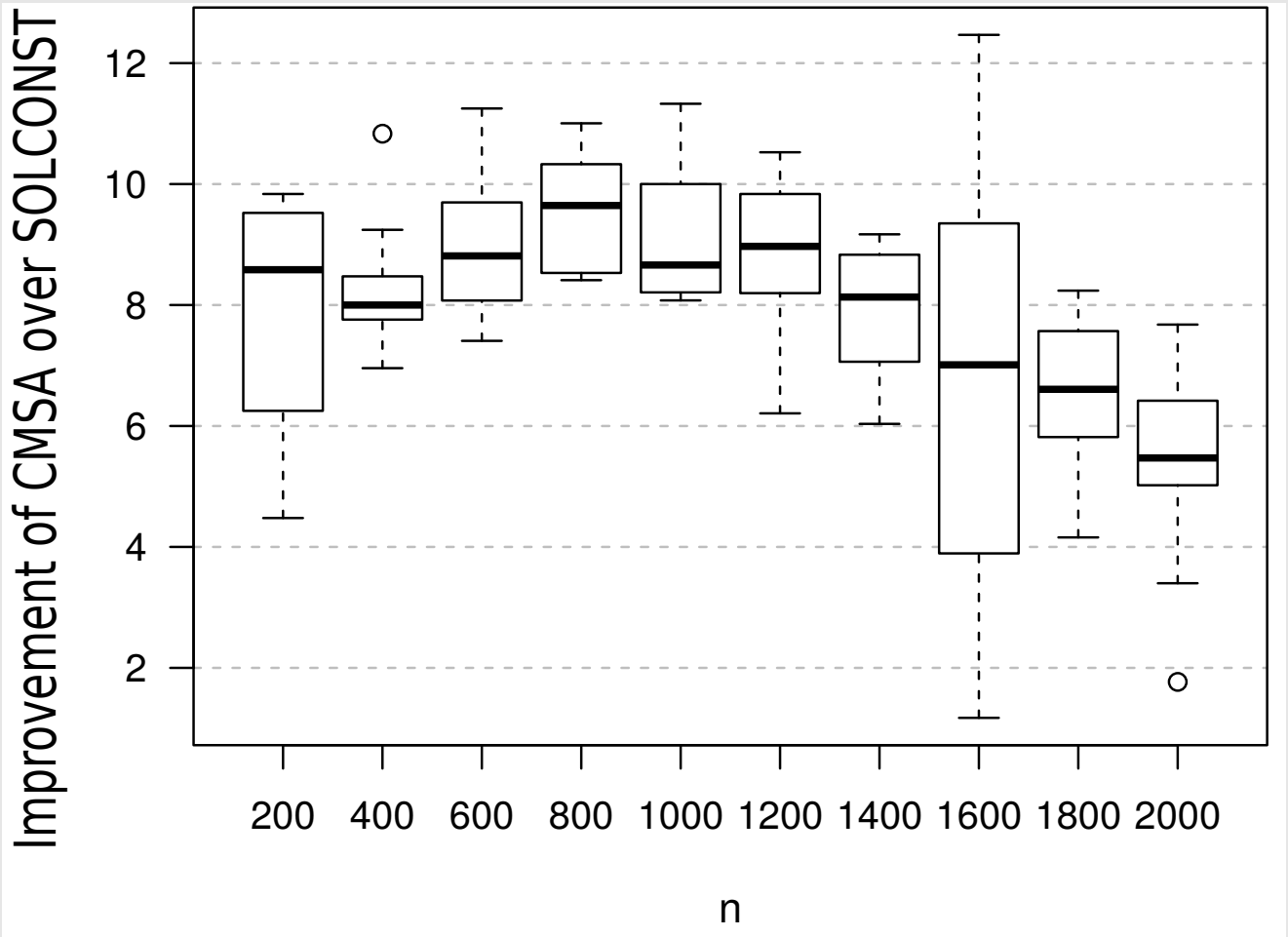
- **In total:** 300 instances
- Input sequence length: $n \in \{200, 400, \ldots, 1800, 2000\}$
- Alphabet size: $|\Sigma| \in \{4, 12, 20\}$

Parameter tuning with `irace`

n	n_a	age_{max}	d_{rate}	l_{size}	t_{max}
400	50	10	0.0	10	60
800	50	10	0.5	10	240
1200	50	10	0.9	10	480
1600	50	5	0.9	10	480
2000	50	10	0.9	10	480

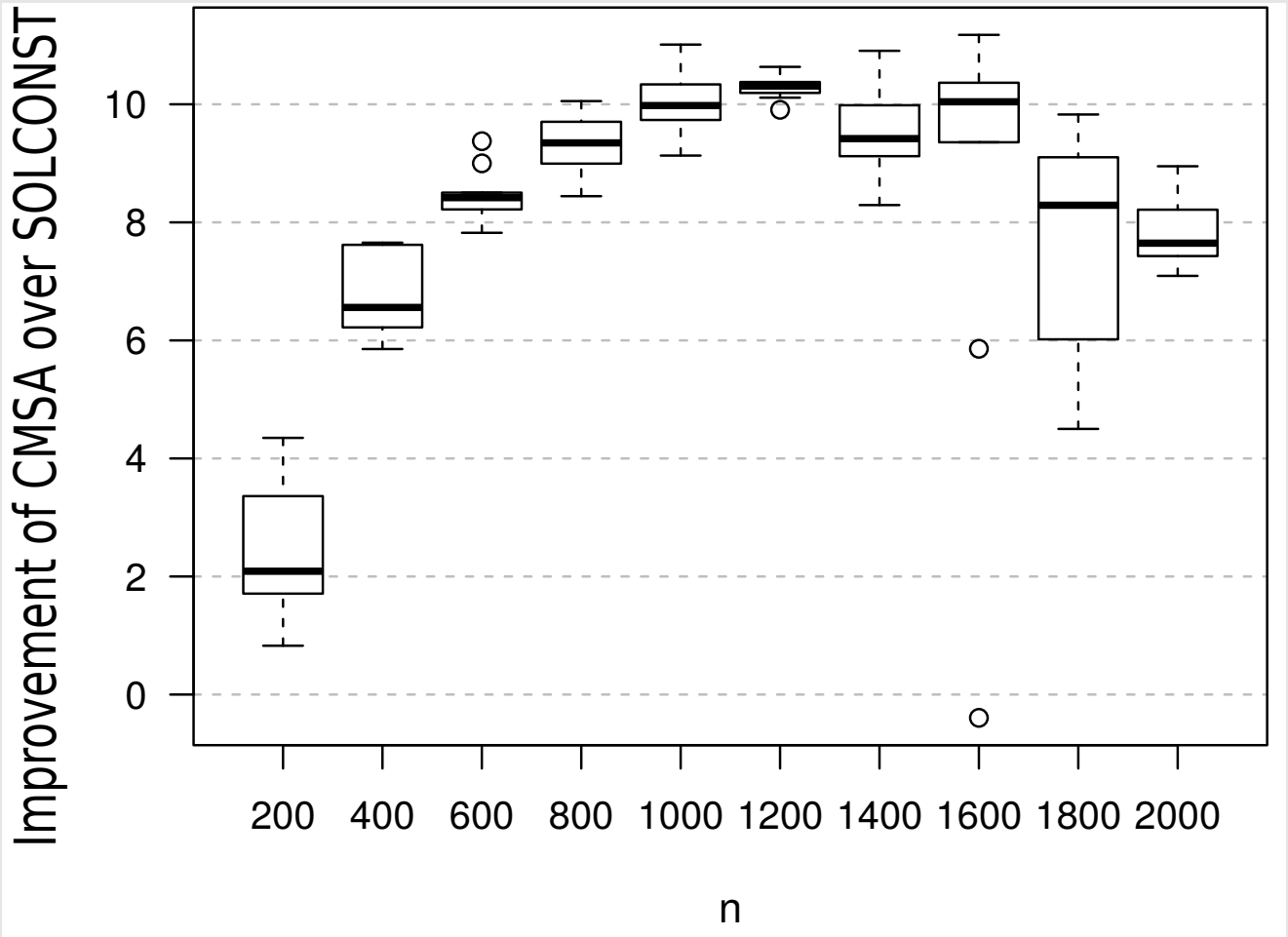
Performance of CMSA (1)

Improvement of CMSA over the greedy heuristic: $|\Sigma| = 4$



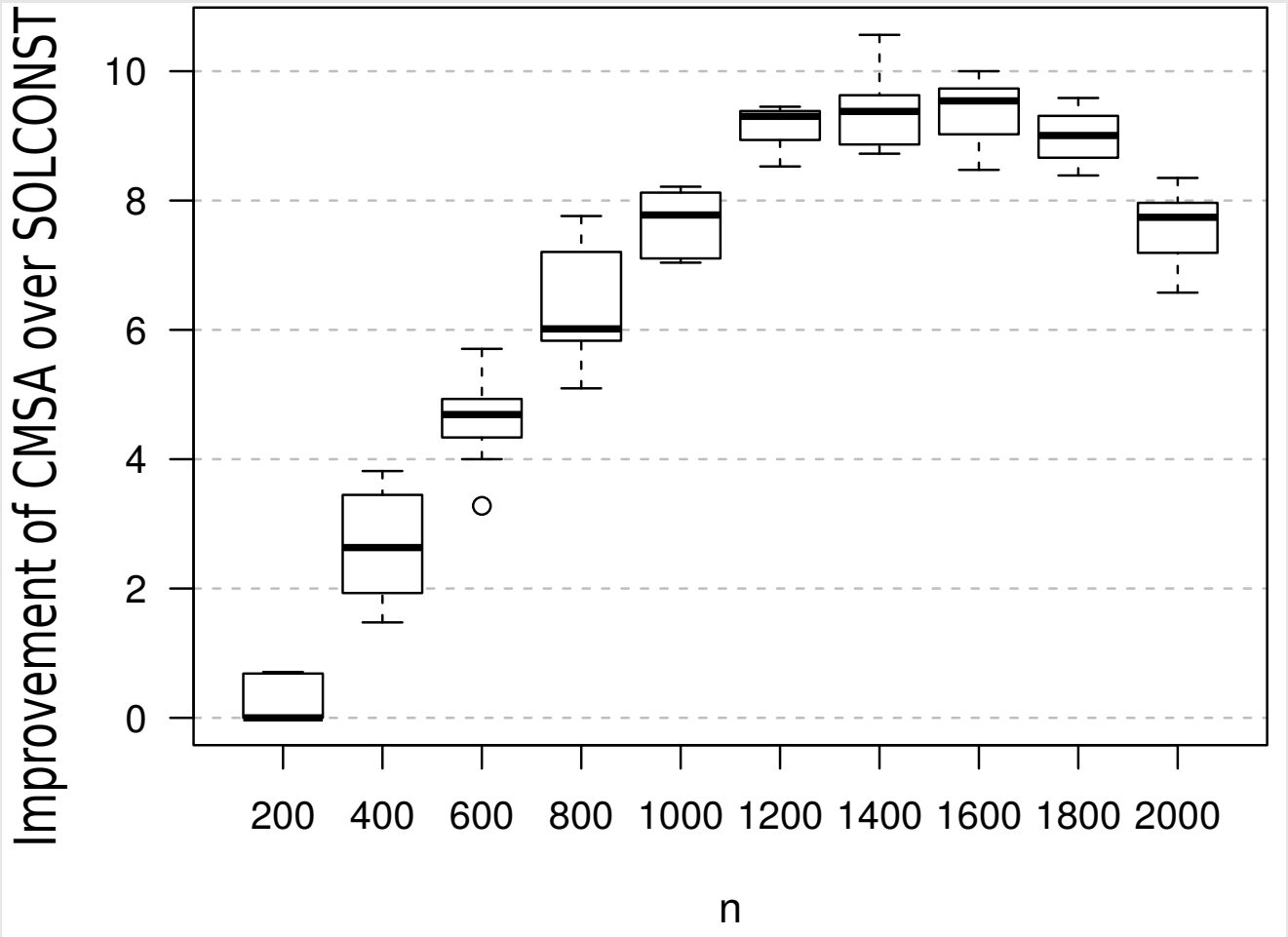
Performance of CMSA (2)

Improvement of CMSA over the greedy heuristic: $|\Sigma| = 12$



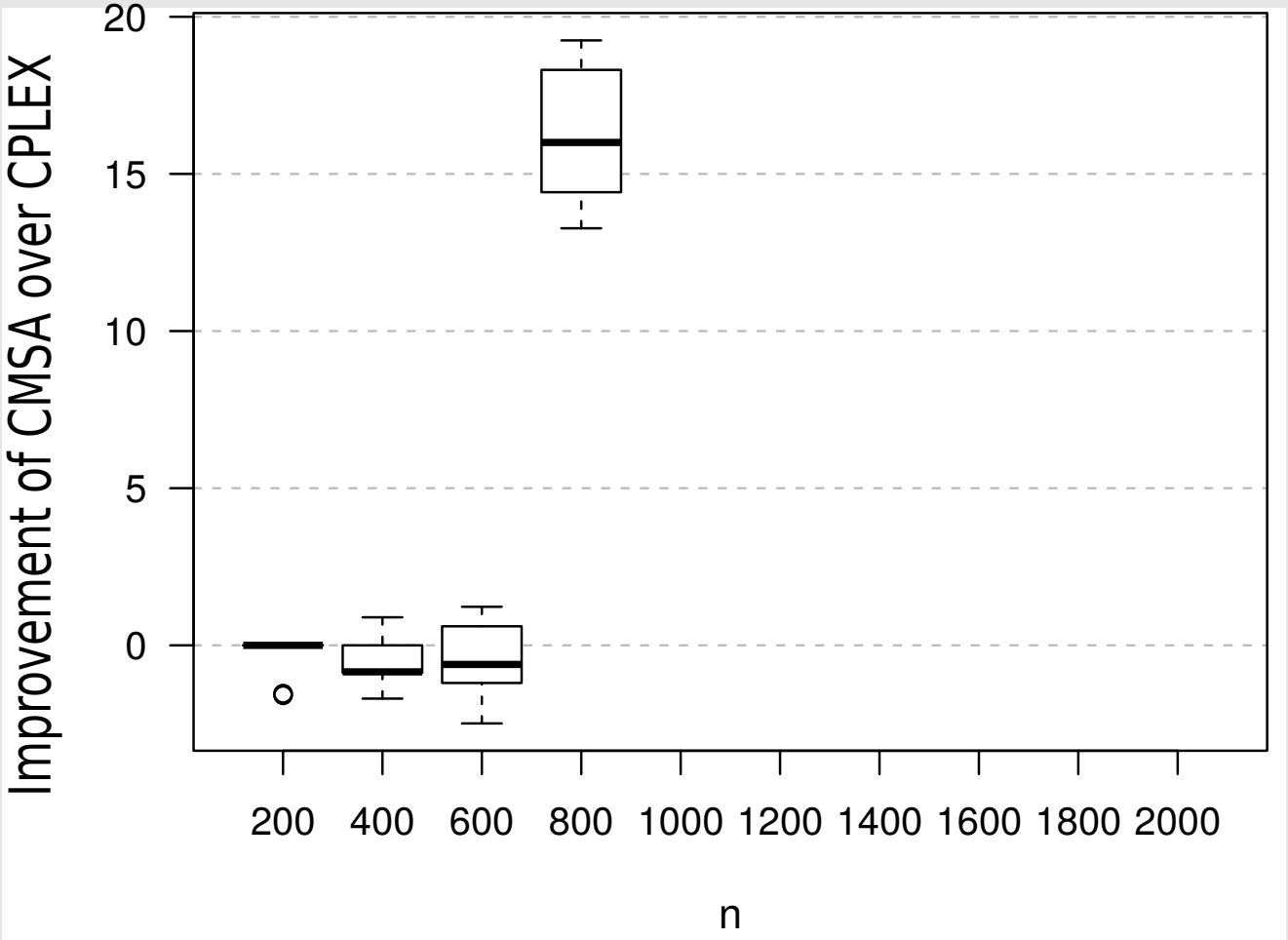
Performance of CMSA (3)

Improvement of CMSA over the greedy heuristic: $|\Sigma| = 20$



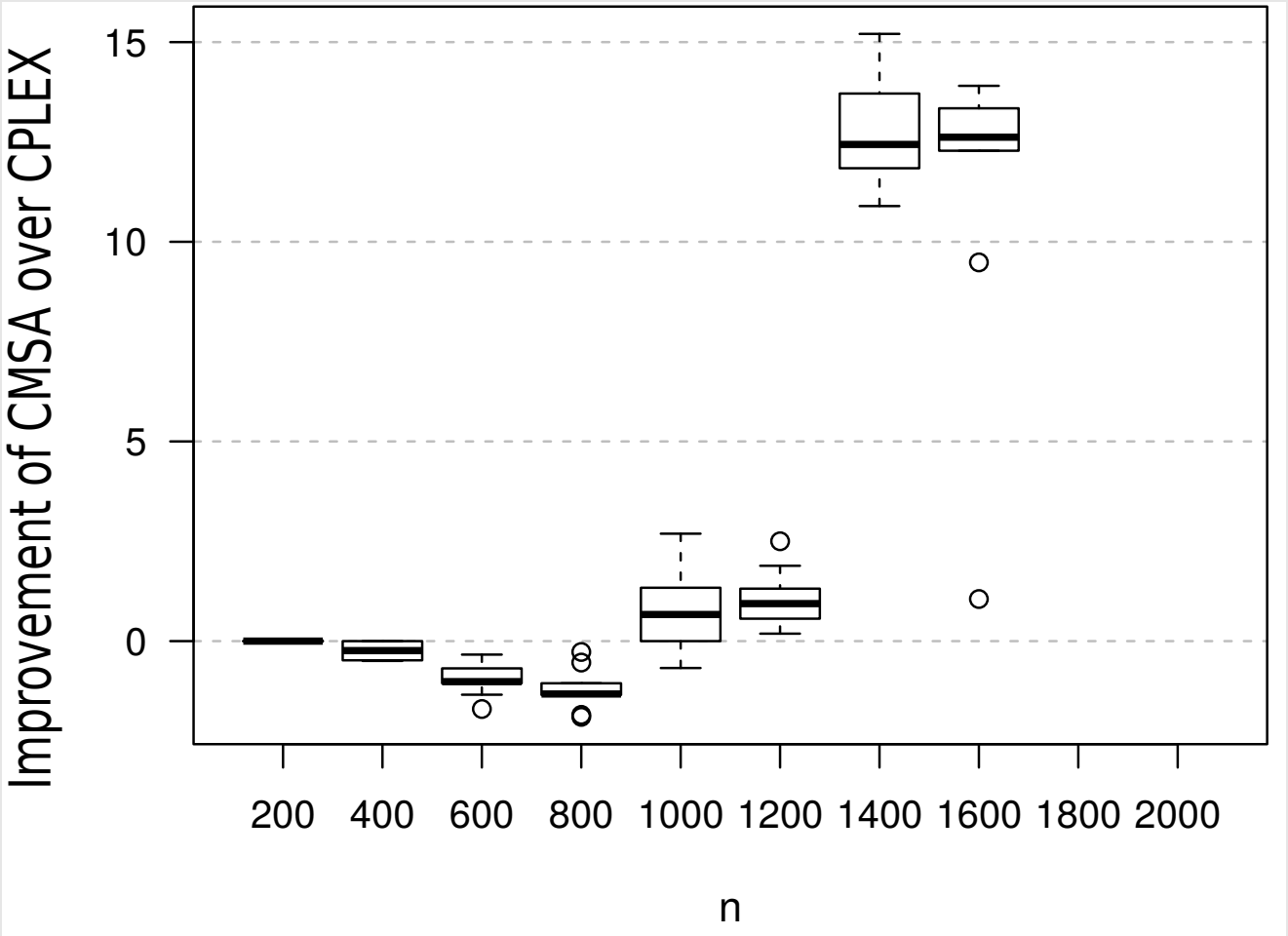
Performance of CMSA (4)

Improvement of CMSA over CPLEX: $|\Sigma| = 4$



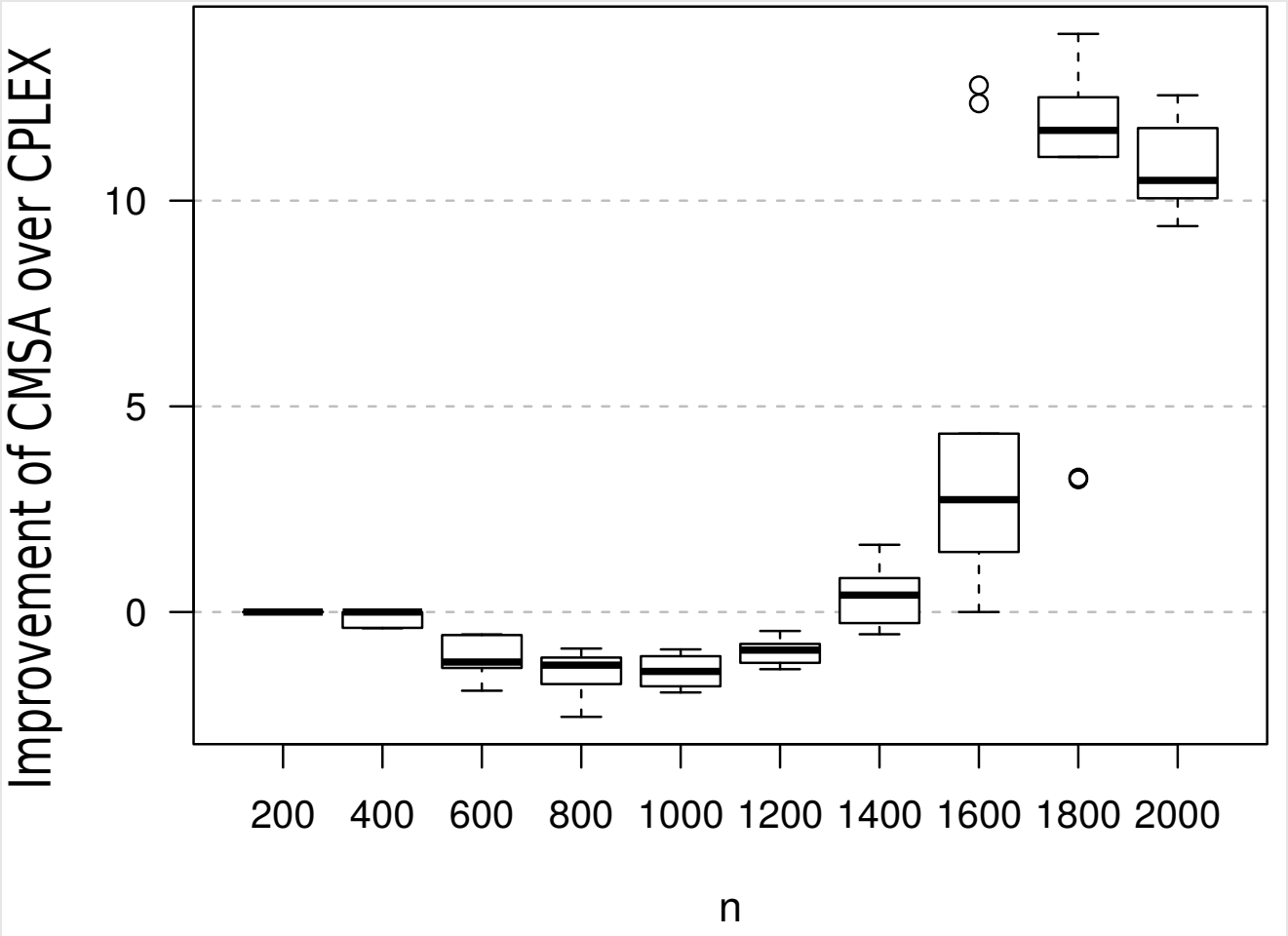
Performance of CMSA (5)

Improvement of CMSA over CPLEX: $|\Sigma| = 12$



Performance of CMSA (6)

Improvement of CMSA over CPLEX: $|\Sigma| = 20$



Questions?

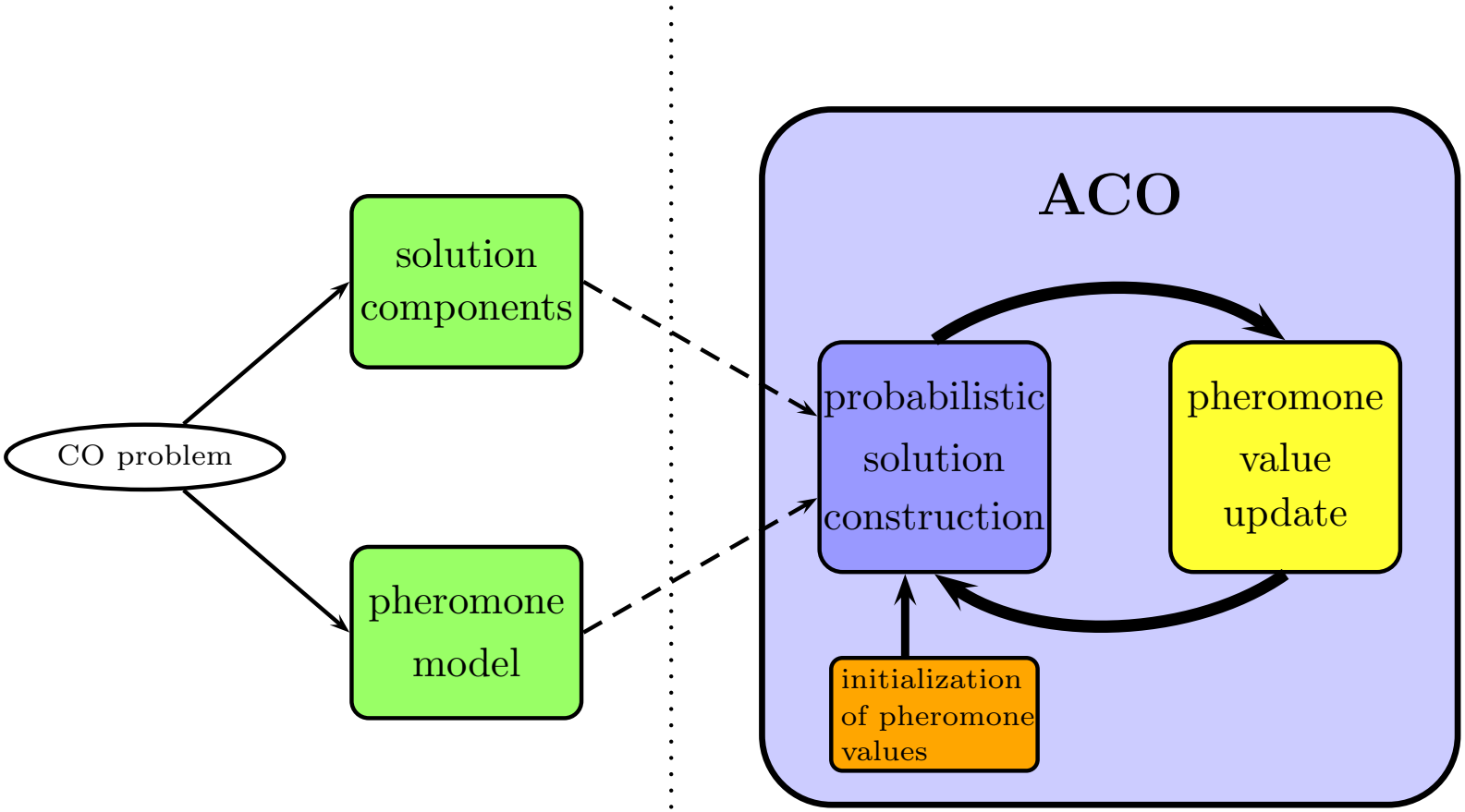
Extending ACO with negative learning

- **Observation:** ACO is based on learning from positive examples
- **Nature shows:** learning from negative examples can be useful

IIIA PhD Student
Teddy Nurcahyadi

ANTS 2020
Best Paper Award

Extending ACO with negative learning



Note

- ACO has been applied to many combinatorial optimization problems
- The type of learning used is positive learning

Negative learning in nature

Observation

Behavior of certain insects is governed not only by reward, but also by penalty / punishment

Examples

- **Pharaoh Ant** (*Monomorium pharaonis*) → mark routes leading to non-profitable food source [Robinson 2005, 2007]
- **Honey Bees** (*Apis mellifera linguistica*) → mark flowers that are already exhausted of nectar [Giurfa 1993]
- **Black garden-ants** (*Lasius niger*) → enables the colony to maintain a flexible foraging system [Grueter 2012]
- **Tsetse flies** (*Glossina morsitans*) and **house flies** (*Musca domestica*) → prevents wasteful activities related to their reproduction behaviour [Schlein 1981]

Previous attempts

Negative learning approaches from the literature

- **[Maniezzo 1999]** and **[Cordón et al. 2000]** were the first ones to make use of an active decrease of pheromone values associated to solution components appearing in low-quality solutions.
- **Montgommery et al. 2002]** proposed the first strategy using explicitly anti-pheromone values.
- **Ramos 2013]** proposed a method that uses a coevolved compromise between positive and negative feedback.
- **Simons et al. 2016]** explored different extensions of the approaches from [Montgommery et al. 2002].
- **Rojas et al. 2018]** proposed a mechanism based on opposite learning.

However

None of these approaches is much used nowadays.

Application to the MDS problem

Baseline algorithm: MMAS

- Negative learning mechanisms is implemented on top of a standard ACO algorithm: $\mathcal{MAX}$-$\mathcal{MIN}$ **Ant System (MMAS) in the hypercube framework** (Blum, Dorigo, 2004).

Main Characteristics of the Baseline Algorithm

1. **At each iteration:** n_a solutions are probabilistically generated both based on pheromone and on greedy information
2. The pheromone values are modified using (at most) three solutions: the best-so-far, the iteration-best and the restart-best solutions
3. Restarts are performed once convergence is detected

Pheromone Model and Constr. Probabilities

Pheromone models

- **Standard pheromones:** A pheromone value τ_i for each vertex v_i of the input graph
- **Negative pheromones:** A pheromone value τ_i^{neg} for each vertex v_i

Solution construction probabilities

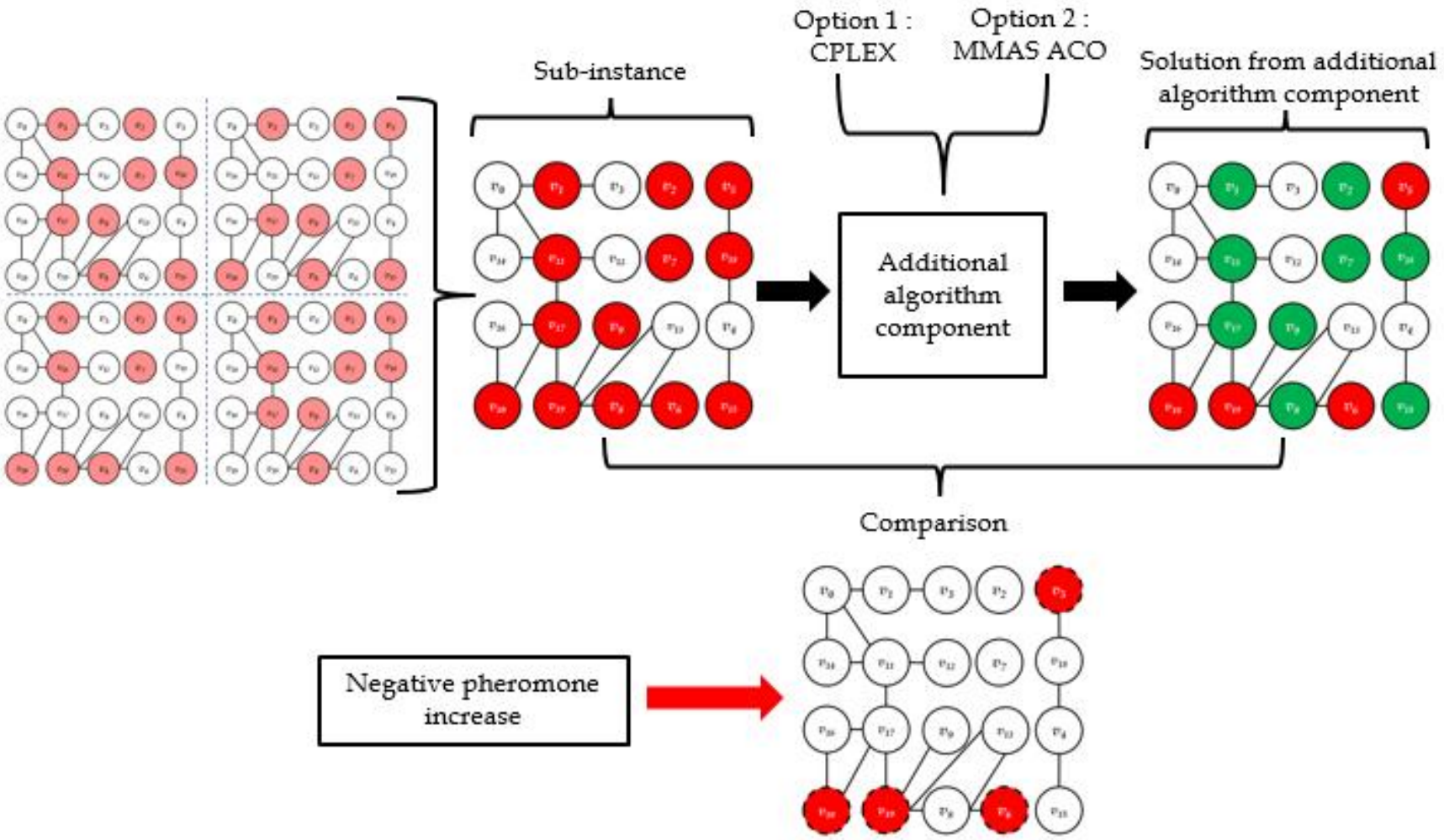
Given a partial solution S

$$\mathbf{p}(v_k \mid S) := \frac{\eta_k \cdot \tau_k \cdot (1 - \tau_k^{\text{neg}})}{\sum_{v_l \in V \setminus S} \eta_l \cdot \tau_l \cdot (1 - \tau_l^{\text{neg}})}$$

Note

$\mathbf{p}(v_k \mid S)$ is the probability to chose v_k as the next vertex to be added to S

ACO + negative learning: main idea

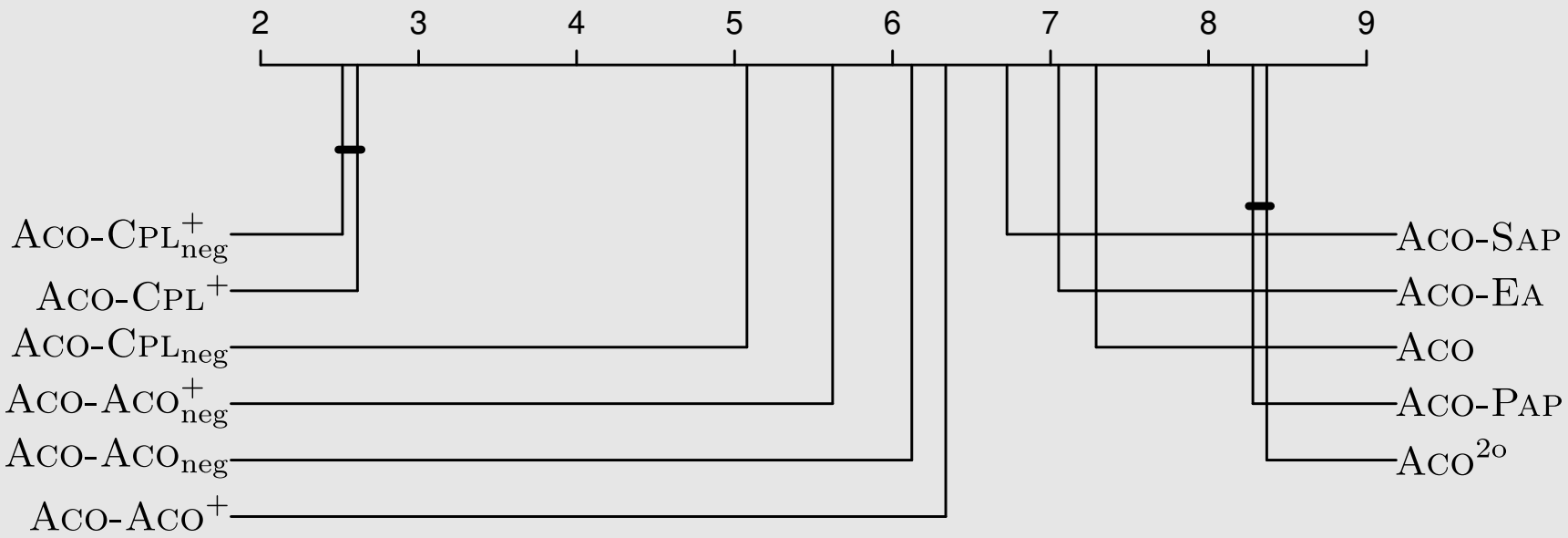


Tested algorithm variants

1. Baseline algorithm:
   - ACO
2. Our negative learning variants:
   - ACO-CPL$^{+}_{\text{neg}}$: negative learning + using CPLEX result for positive learning
   - ACO-CPL$_{\text{neg}}$: negative learning
   - ACO-CPL^{+}: only using CPLEX result for positive learning
   - ACO-ACO$^{+}_{\text{neg}}$
   - ACO-ACO$_{\text{neg}}$
   - ACO-ACO^{+}
3. Negative learning variants from the literature:
   - ACO-SAP, ACO-PAP, ACO-EA, ACO^{2o}

Global results for 160 MDS instances

Critical difference (CD) plot



Observations

- Our variants outperform the baseline ACO and the other competitors
- The ACO-CPL variants perform better than the ACO-ACO variants

Comparison to the state-of-the-art

Comparison

Instance family	CC^{2}FS	FastMWDS	RLS$_o$	ScBppw	FastDS	Aco-Cpl$^+_{\text{neg}}$
V50U150	**12.9**	**12.9**	**12.9**	**12.9**	**12.9**	**12.9**
V50U200	**9.4**	**9.4**	**9.4**	**9.4**	**9.4**	**9.4**
V100U150	**17.0**	**17.0**	**17.0**	17.3	**17.0**	**17.0**
V100U200	**10.4**	**10.4**	**10.4**	10.6	**10.4**	**10.4**
V250U150	**18.0**	**18.0**	**18.0**	19.0	**18.0**	**18.0**
V250U200	**10.8**	**10.8**	**10.8**	11.5	**10.8**	**10.8**
V500U150	**18.5**	**18.5**	18.6	20.1	**18.5**	**18.5**
V500U200	**11.2**	**11.2**	**11.2**	12.4	**11.2**	**11.2**
V800U150	**19.0**	**19.0**	19.1	20.9	**19.0**	**19.0**
V800U200	**11.7**	**11.7**	11.9	12.6	11.8	**11.7**
V1000U150	**19.1**	**19.1**	19.2	21.3	**19.1**	**19.1**
V1000U200	**12.0**	**12.0**	**12.0**	13.0	**12.0**	**12.0**

Competitors

- CC^{2}FS ([Wang, Cai, and Yin, 2017])
- FastMWDS ([Wang, Cai, Chen et al., 2018])
- RLS$_o$ ([Chalupa, 2018])
- ScBppw ([Fan et al., 2019])
- FastDS ([Cai et al., 2020])

Questions?

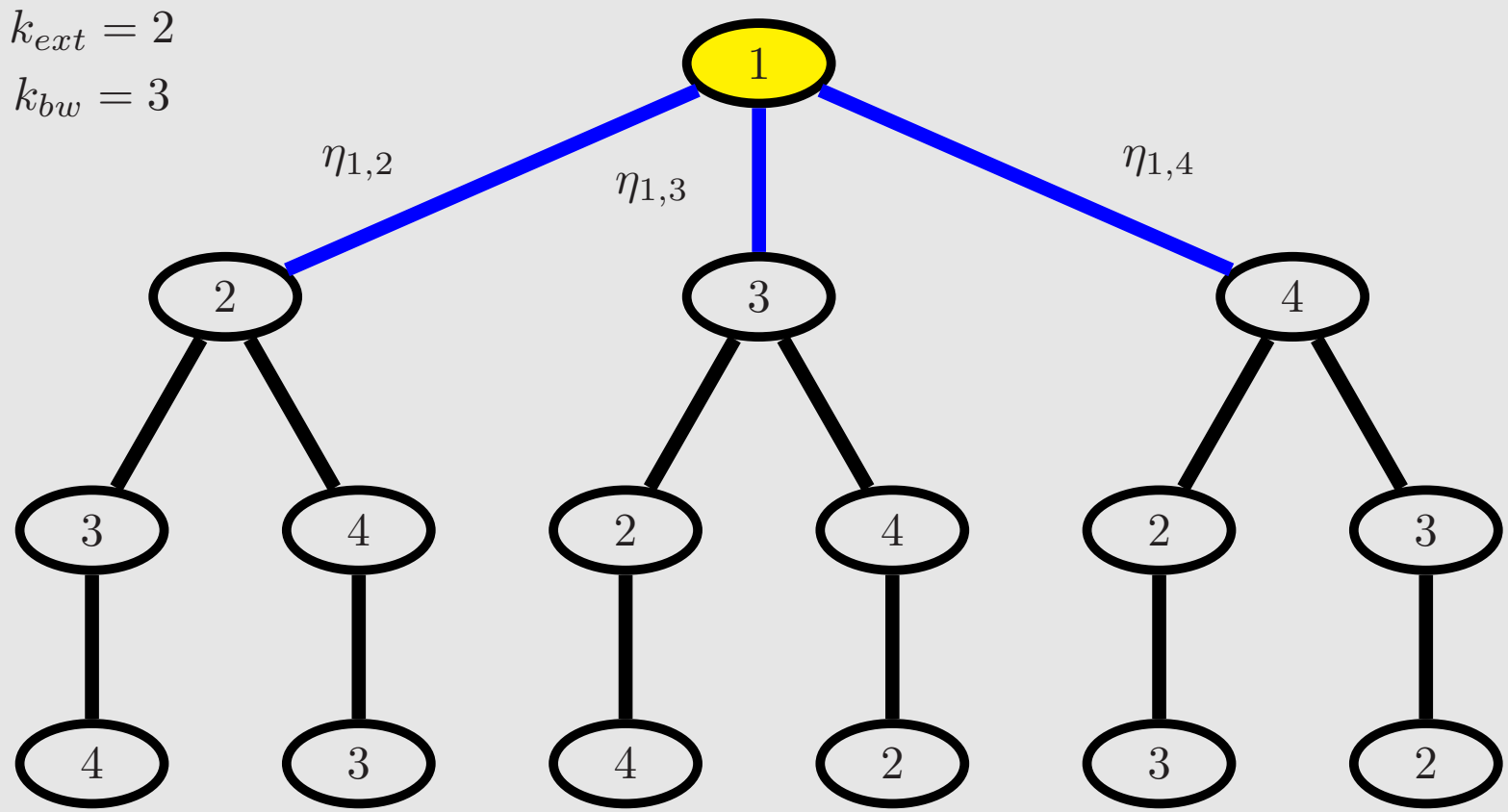
Hybridization example: Beam-ACO

What is Beam-ACO?

- Beam-ACO is a hybrid algorithm that results from the combination of **ant colony optimization** with a tree search method called **beam search**.
- **Main publication:** Blum, C. (2005). Beam-ACO, Hybridizing ant colony optimization with beam search: An application to open shop scheduling. Computers & Operations Research, 32(6), 1565-1591.

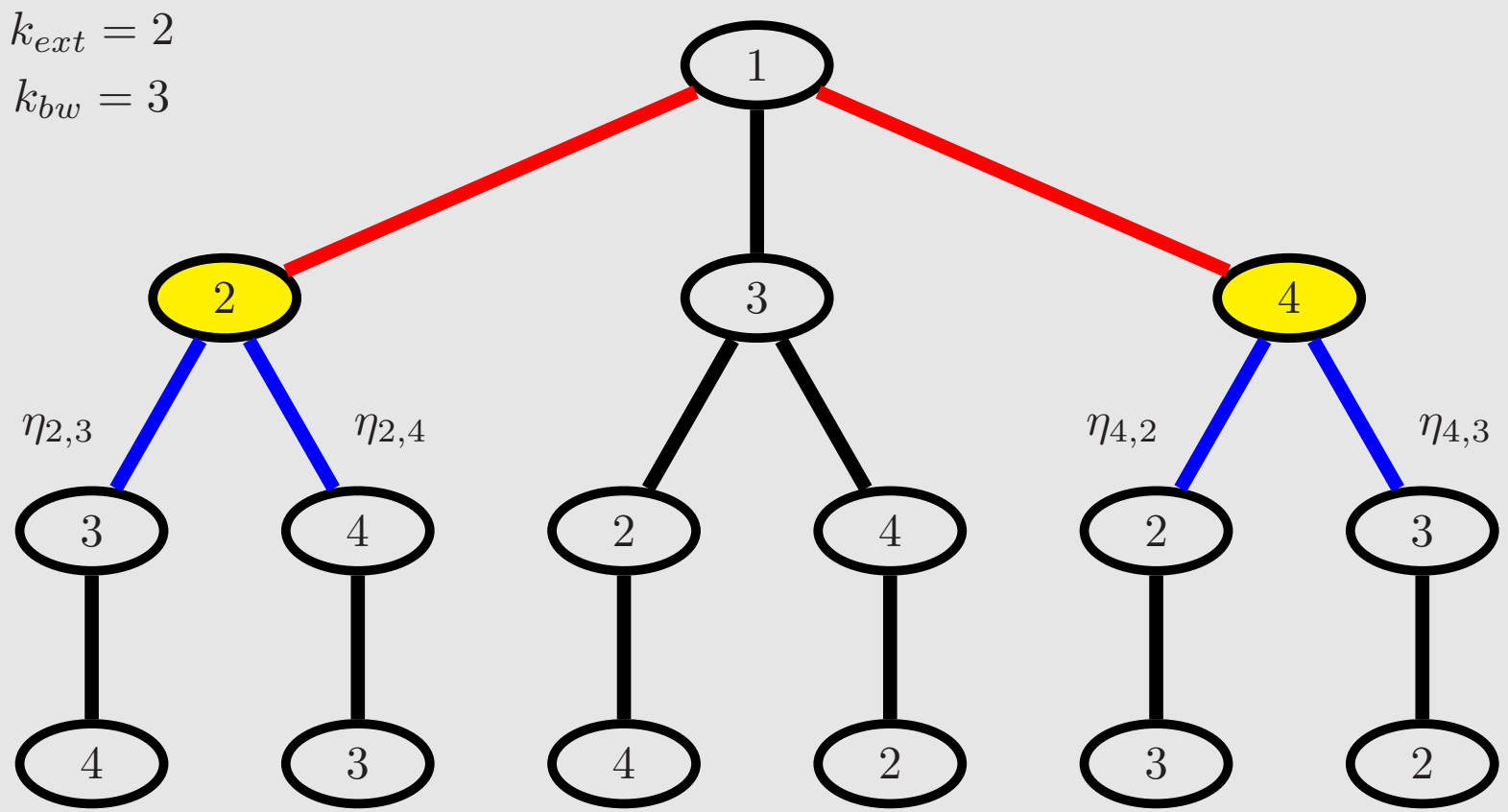
Beam search for the TSP (1)

First construction step



Beam search for the TSP (2)

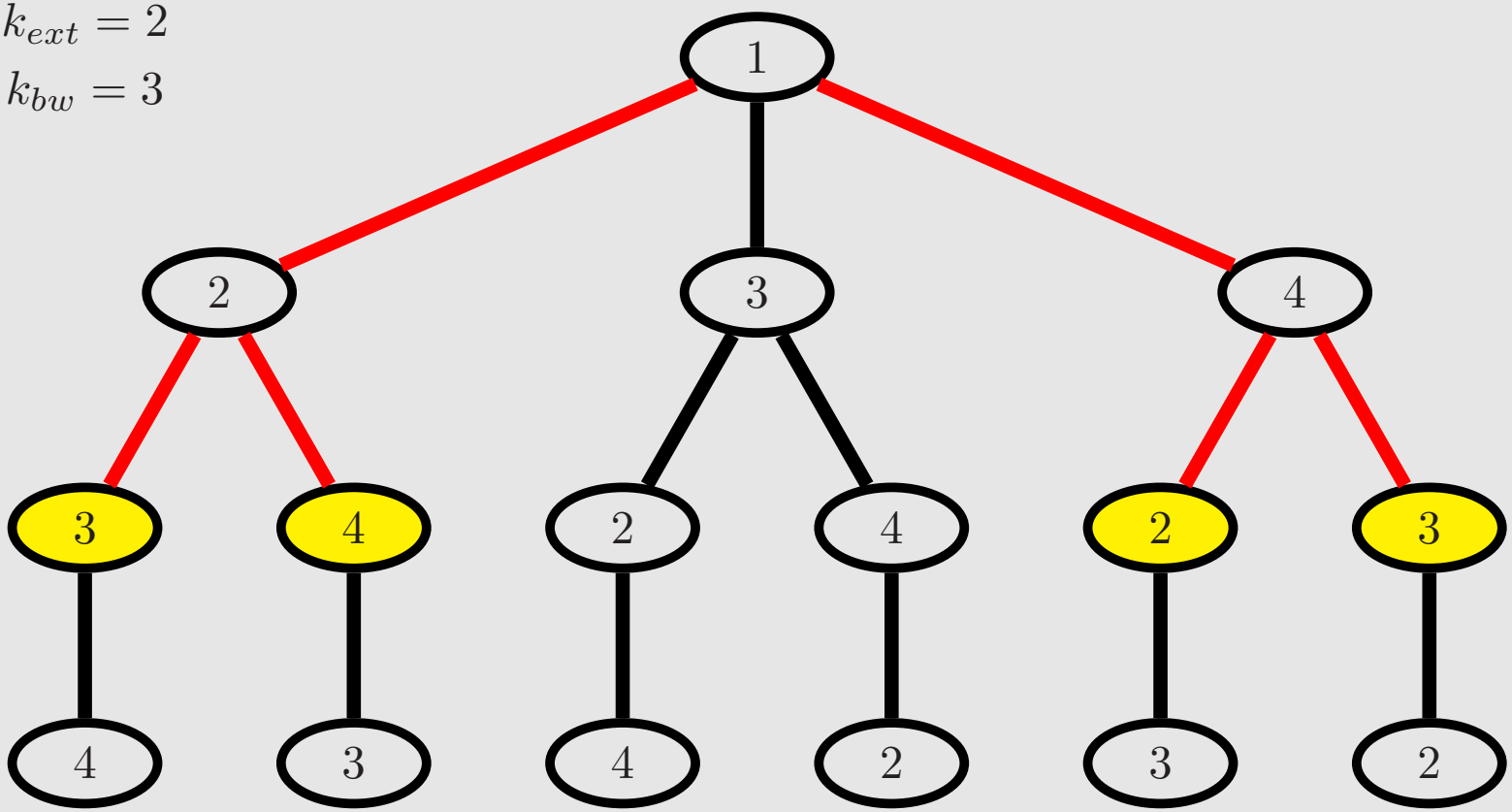
Second construction step



Beam search for the TSP (3)

After the 2nd construction step: use of a lower bound

$k_{ext} = 2$
$k_{bw} = 3$

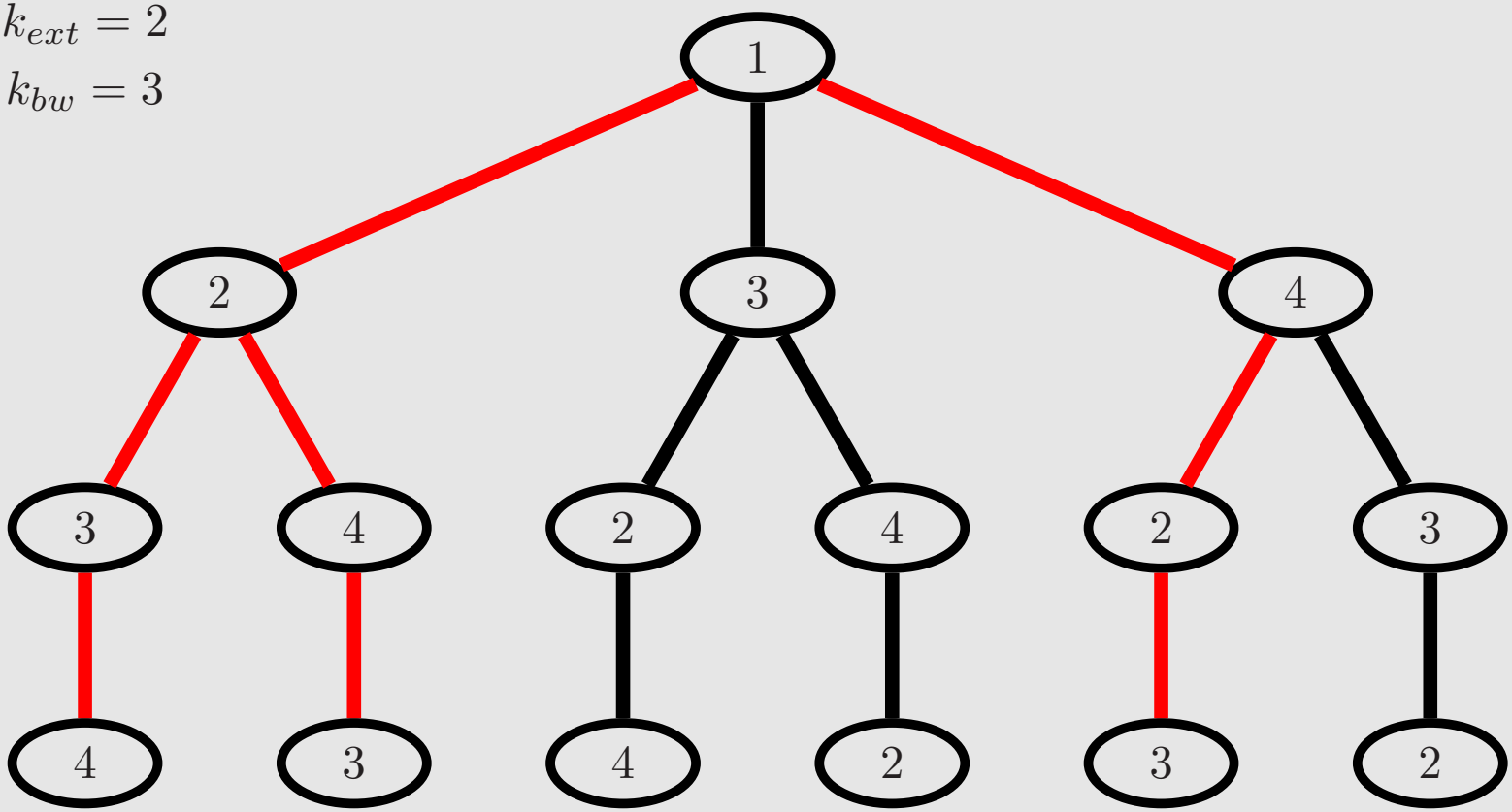


Beam search for the TSP (4)

Third construction step

$k_{ext} = 2$

$k_{bw} = 3$



Beam-ACO: basic idea

Basic idea of Beam-ACO

- Instead of n_a independent solution constructions per iteration: perform a probabilistic beam search with beam width $k_{bw} = n_a$.
- The probabilities for the construction steps are calculated on the basis of (1) the greedy function and (2) the pheromone values, just like in ACO.

Advantages of Beam-ACO

- Strong heuristic guidance by a lower bound
- Embedded in the adaptive ACO framework
- Beam-ACO utilizes two types of complementary problem knowledge (heuristic + lower bound)

Beam-ACO: applications

Applications of Beam-ACO (a selection)

- **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)

Questions?