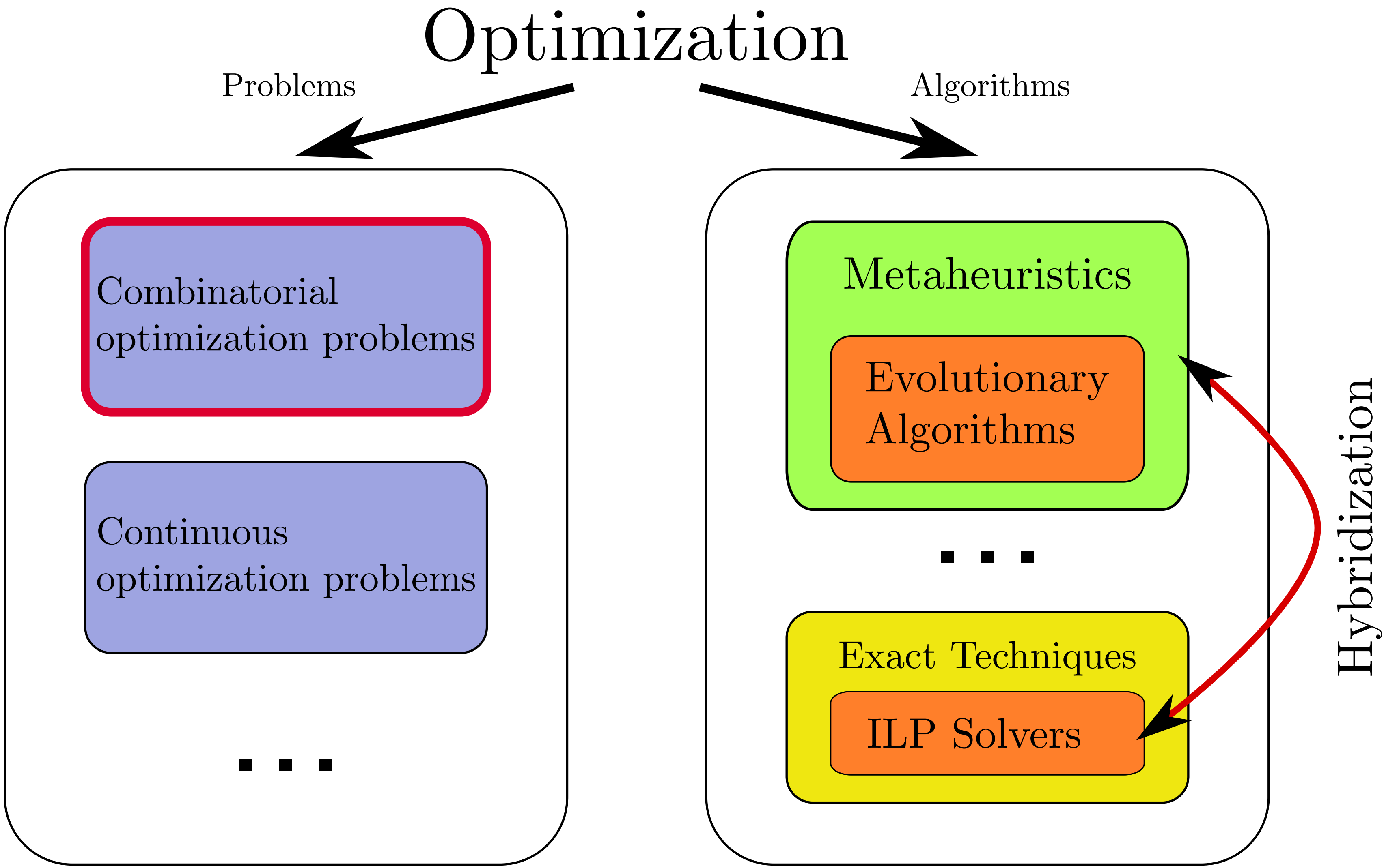
Combining Metaheuristics with ILP Solvers in Combinatorial Optimization

Christian Blum

University Of The Basque Country

Ikerbasque, Basque Foundation For Science

Preliminaries: Preparing the Grounds



Motivation and Outline (1)

Motivation:

- 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 nothing like that is known
  - ⋆ We only have very few generally applicable techniques
  - ⋆ We do not really know for which type of problem they work well

Motivation and Outline (2)

Outline:

- **Short Intro:** Hybrid metaheuristics
- **How to combine metaheuristics with ILP solvers?**
  - **Standard method:** **Large neighborhood search (LNS)**
- **Hypothesis** about the conditions in which LNS does NOT work
- **What can we use instead of LNS?**
  - **New hybrid:** **Construct, Merge, Solve & Adapt (CMSA)**

Hybrid Metaheuristics

Short introduction

Hybrid metaheuristics: definition

Definition: What is a hybrid metaheuristic?

- Problem: a precise definition is not possible

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

Specific topic of this presentation

Specific topic today: Combination between metaheuristics and ILP Solvers

Available general purpose ILP solvers:

- IBM ILOG CPLEX: free for academic purposes
- Gurobi: free for academic purposes
- FICO Xpress: 30 days free trial. Restricted student version for free
- MOSEK: free for academic purposes

Example of an integer linear program (1)

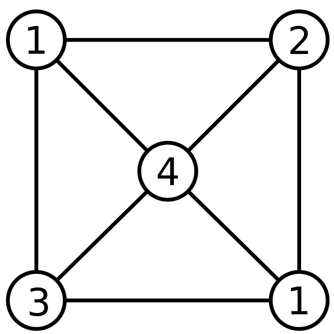
Optimization problem: **Minimum weight dominating set** (MWDS)

- **Given:** An undirected graph $G = (V, E)$; each $v_i \in V$ has a weight $w(v_i) \geq 0$
- **Valid solutions:** Any subset $S \subseteq V$ is a valid solution iff

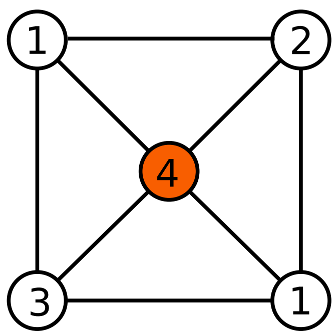
$$\forall\ v_i \in V\colon\ N[v_i] \cap S \neq \emptyset$$

- **Optimization goal:** Find a solution S^* that minimizes

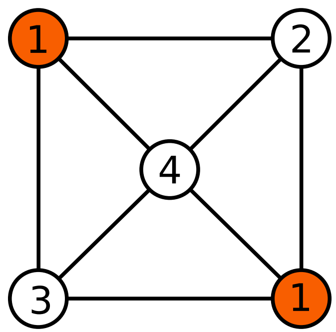
$$f(S^*) := \sum_{v_i \in S^*} w(v_i)$$



Given graph G



A valid solution



The optimal solution

Example of an integer linear program (2)

Stating the MWDS problem as an ILP:

$$\min \sum_{v_i \in V} w(v_i) \cdot x_i \tag{1}$$

subject to:

$$\sum_{v_j \in N[v_i]} x_j \geq 1 \quad \text{for } v_i \in V \tag{2}$$

$$x_i \in \{0, 1\} \quad \text{for } v_i \in V$$

Beware:

- In this ILP: linear number of variables and constraints
- Any problem may be expressed as an ILP in various different ways

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

ILP-based Large Neighborhood Search

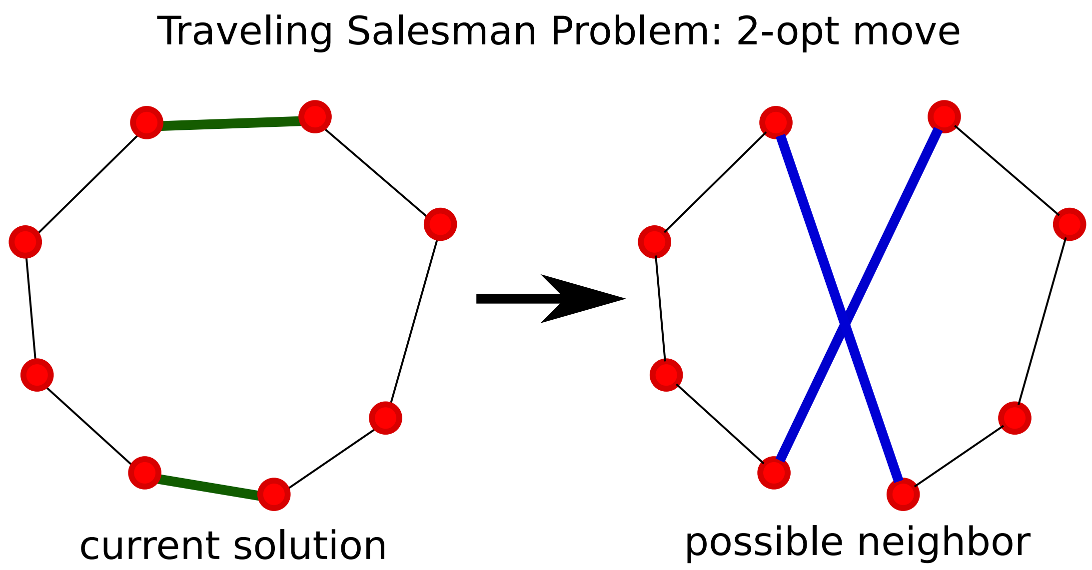
Principle: Use of an ILP solver for finding the best neighbor in a large neighborhood of a solution

David Pisinger, Stefan Ropke. **Large Neighborhood Search**, *Handbook of Metaheuristics*, International Series in Operations Research & Management Science Volume 146, 2010, pp 399-419

Neighborhood search (1)

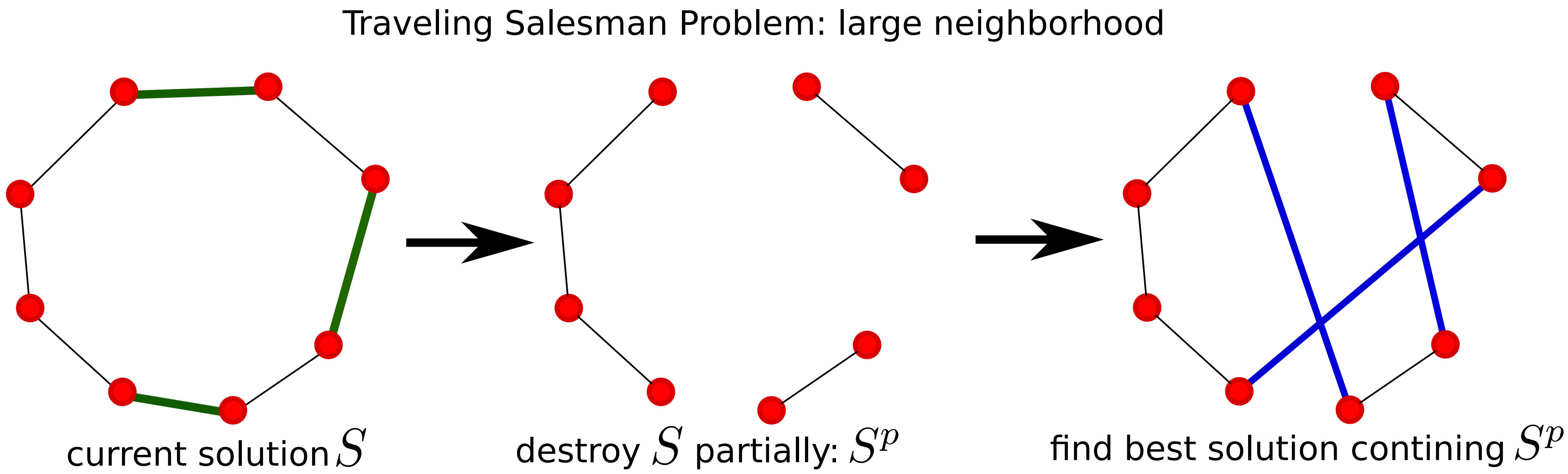
- Crucial ingredient of neighborhood search: Choice of a neighborhood
- Usual in metaheuristics based on neighborhood search: rather small neighborhoods

$\boxed{\text{Example of a small neighborhood:}}$ 2-opt neighborhood for the TSP. Each solution has $O(n^2)$ neighbors.



Neighborhood search (2)

Example of a large neighborhood: in the context of the TSP



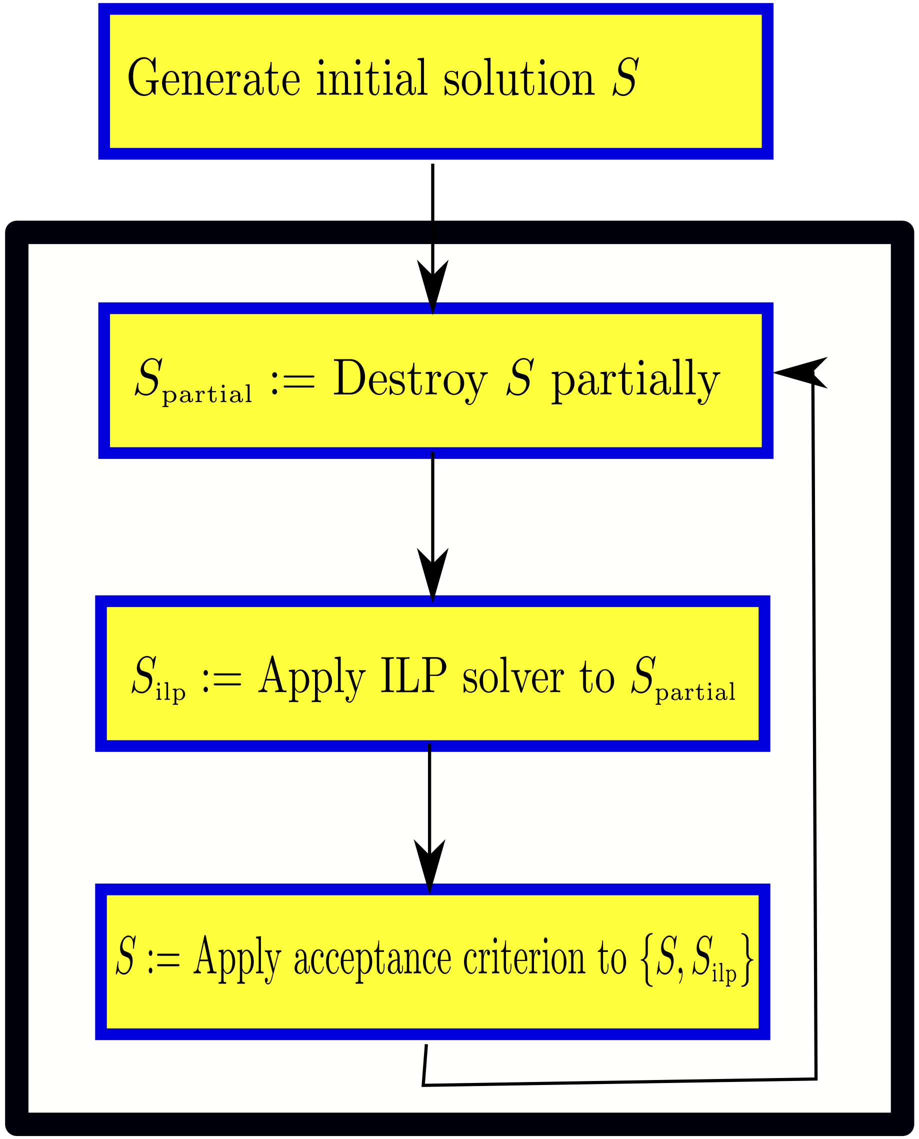
General tradeoff in neighborhood search

- 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

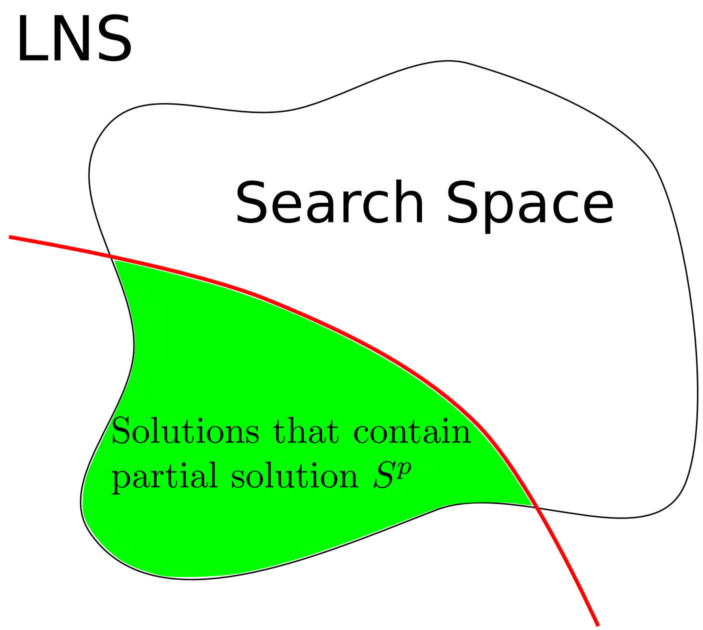
ILP-based large neighborhood search: ILP-LNS



Crucial aspect of ILP-LNS

- Important: Applying an ILP solver to find the best solution containing a specific partial solution S_{partial} means applying the ILP to a reduced search space.

- Consequence: In comparison to the ILP solver, ILP-LNS can be applied to much bigger problem instances



Application example: Minimum weight dominating set

Original ILP:

$$\textbf{min} \sum_{v_i \in V} w(v_i) \cdot x_i$$

subject to:

$$\sum_{v_j \in N[v_i]} x_j \geq 1 \quad \text{for } v_i \in V$$

$$x_i \in \{0,1\} \quad \text{for } v_i \in V$$

How to search for the best solution containing S_{partial}?

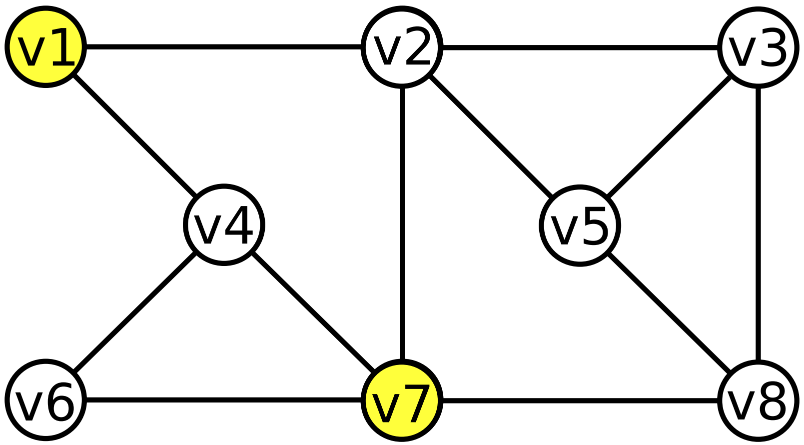
Adding the following constraints: $x_i = 1 \qquad \text{for } v_i \in S_{\text{partial}}$

Generating the initial solution: GREEDY (1)

Definitions:

- V_{cov}: set of covered nodes (w.r.t. a partial solution S)
- $d(v|V_{\text{cov}})$: current degree of v only considering covered nodes

Example:



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

Generating the initial solution: Greedy (2)

Pseudo-code of Greedy:

input: a graph $G = (V, E)$ with node weights

$S := \emptyset$

$V_{\text{cov}} := \emptyset$

while $V_{\text{cov}} \neq V$ **do**

  $v^* := \text{argmax}_{v \in V \setminus V_{\text{cov}}} \left\{ \frac{d(v|V_{\text{cov}})}{w(v)} \right\}$

  $S := S \cup \{v^*\}$

  $V_{\text{cov}} := V_{\text{cov}} \cup N[v^*]$

end while

output: S

Partial destruction of a solution

General principle: removing a certain percentage $perc_{\mathrm{dest}}$ of the nodes in S_{cur}

How to select nodes to be removed?

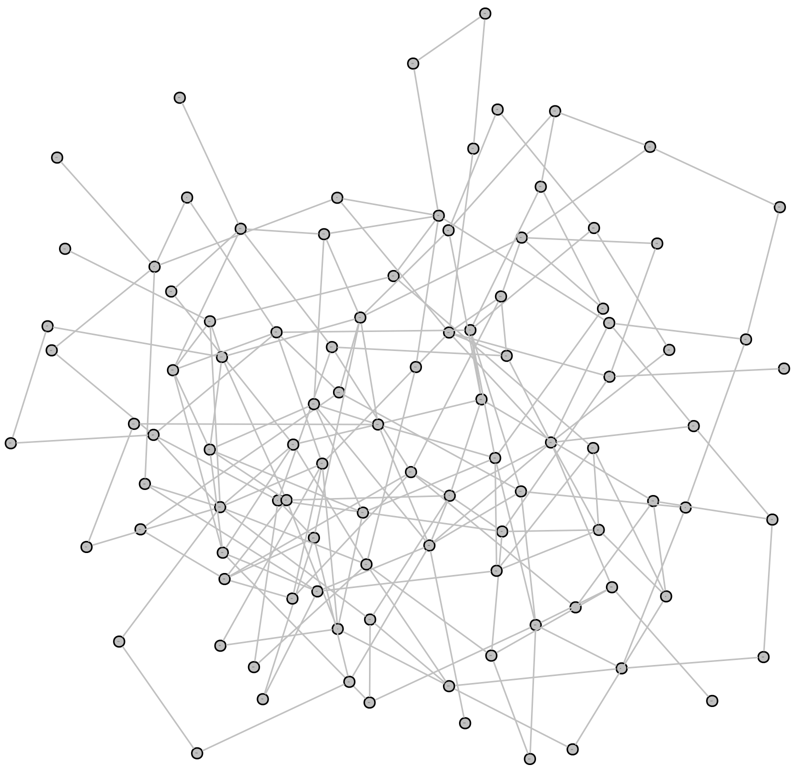
- Destruction type $type_{\mathrm{dest}}$ = random: nodes are chosen uniformly at random
- Destruction type $type_{\mathrm{dest}}$ = heuristically guided): node choice biased by greedy function

Choice of value for $perc_{\mathrm{dest}}$:

- A value is chosen dynamically from $[perc^{l}_{\mathrm{dest}}, perc^{u}_{\mathrm{dest}}]$
- Initially $perc_{\mathrm{dest}} := perc^{l}_{\mathrm{dest}}$
- When no better solution found: $perc_{\mathrm{dest}} := perc_{\mathrm{dest}} + 5$
- When better solution found of upper bound reached: $perc_{\mathrm{dest}} := perc^{l}_{\mathrm{dest}}$

Benchmark instances

- 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 $|V| > 100$: $e_p \in \{0.01, 0.03, 0.05\}$



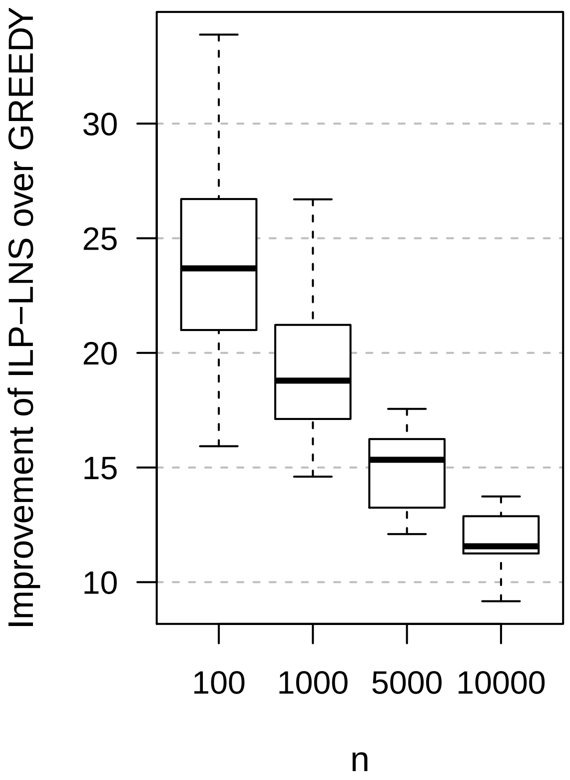
Tuning of ILP-LNS

- $type_{\text{dest}}$ can be random or heuristically guided
- Lower and upper bound $(perc^{l}_{\text{dest}}, perc^{u}_{\text{dest}})$ 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 each application of the ILP solver

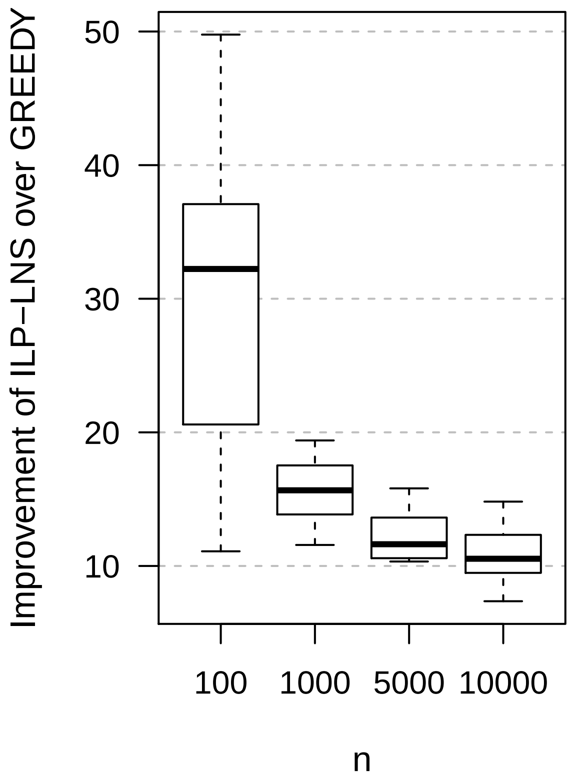
Selected values after tuning with irace:

$\lvert V \rvert$	$type_{\text{dest}}$	$(perc^{l}_{\text{dest}}, perc^{u}_{\text{dest}})$	$t_{\max}$
100	1	$(60, 60)$	2.0
1000	0	$(90, 90)$	10.0
5000	1	$(50, 50)$	5.0
10000	1	$(40, 40)$	10.0

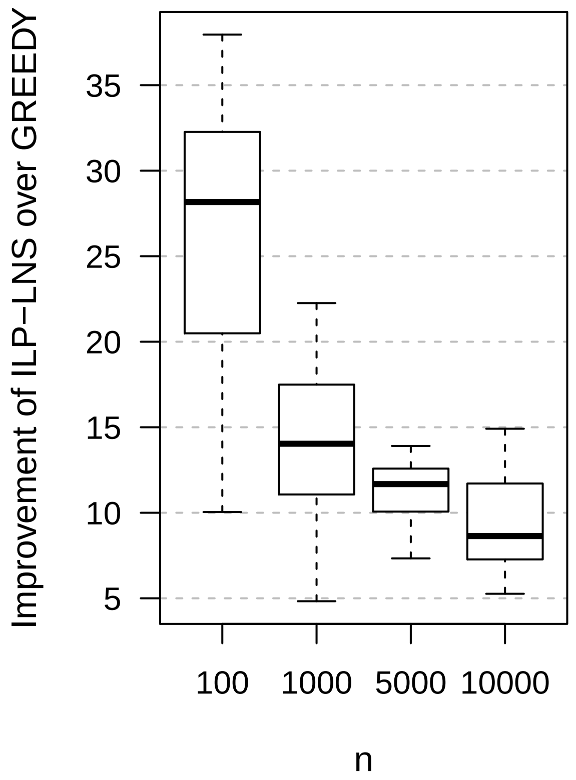
MWDS results: improvement over GREEDY (in percent)



Low density

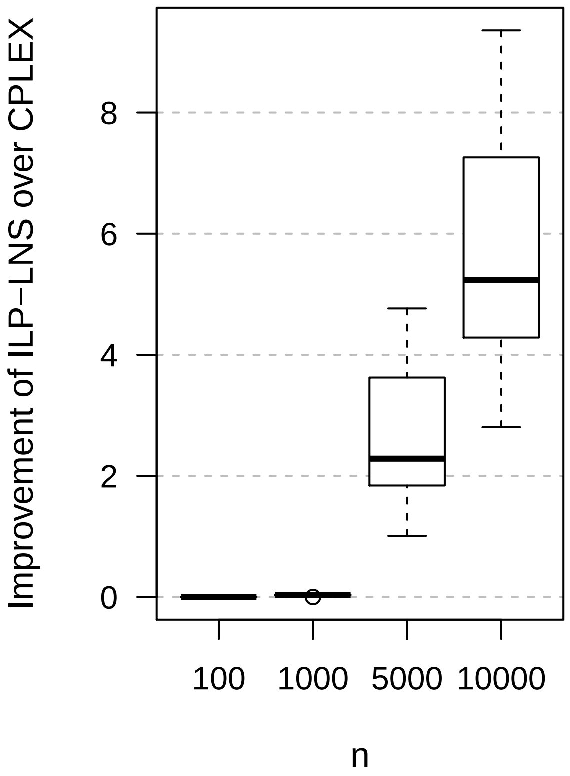


Medium density

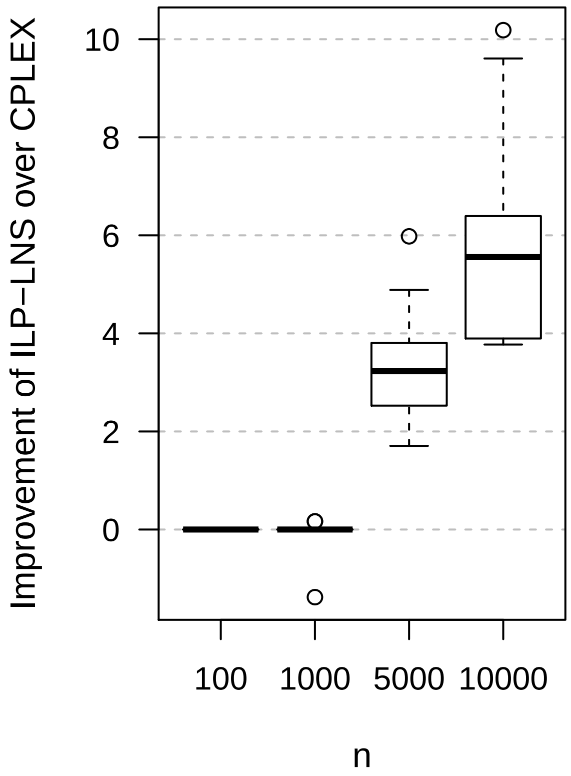


High density

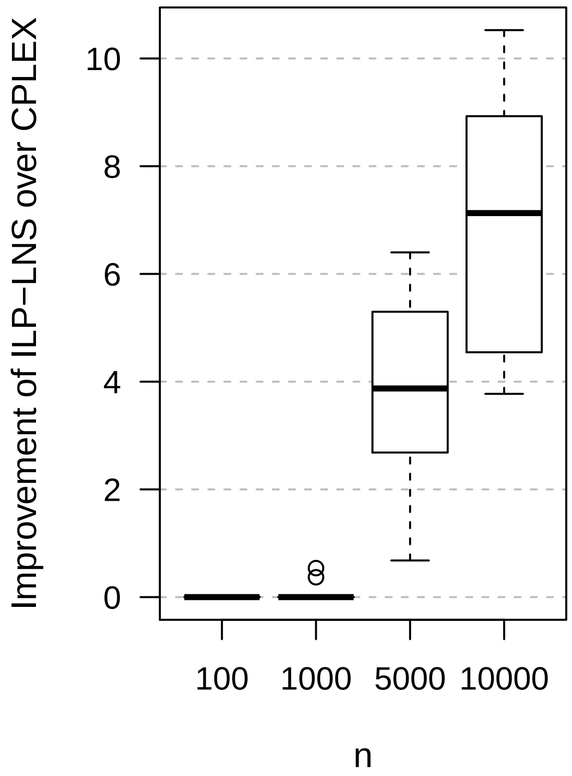
MWDS results: improvement over CPLEX (in percent)



Low density



Medium density



High density

Hypothesis and subsequent question

Hypothesis after studying the LNS literature:

LNS works especially well when the number of solution components (variables) is linear concerning the input parameters of the tackled problem

Question:

What can we do when the ILP of our tackled problem has a large number of solution components (variables)???

Construct, Merge, Solve & Adapt

Principle: Exact solution of sub-instances obtained by joining solutions

Christian Blum, Borja Calvo. **A matheuristic for the minimum weight rooted arborescence problem**. *Journal of Heuristics*, 21(4): 479-499 (2015)

Principal Idea

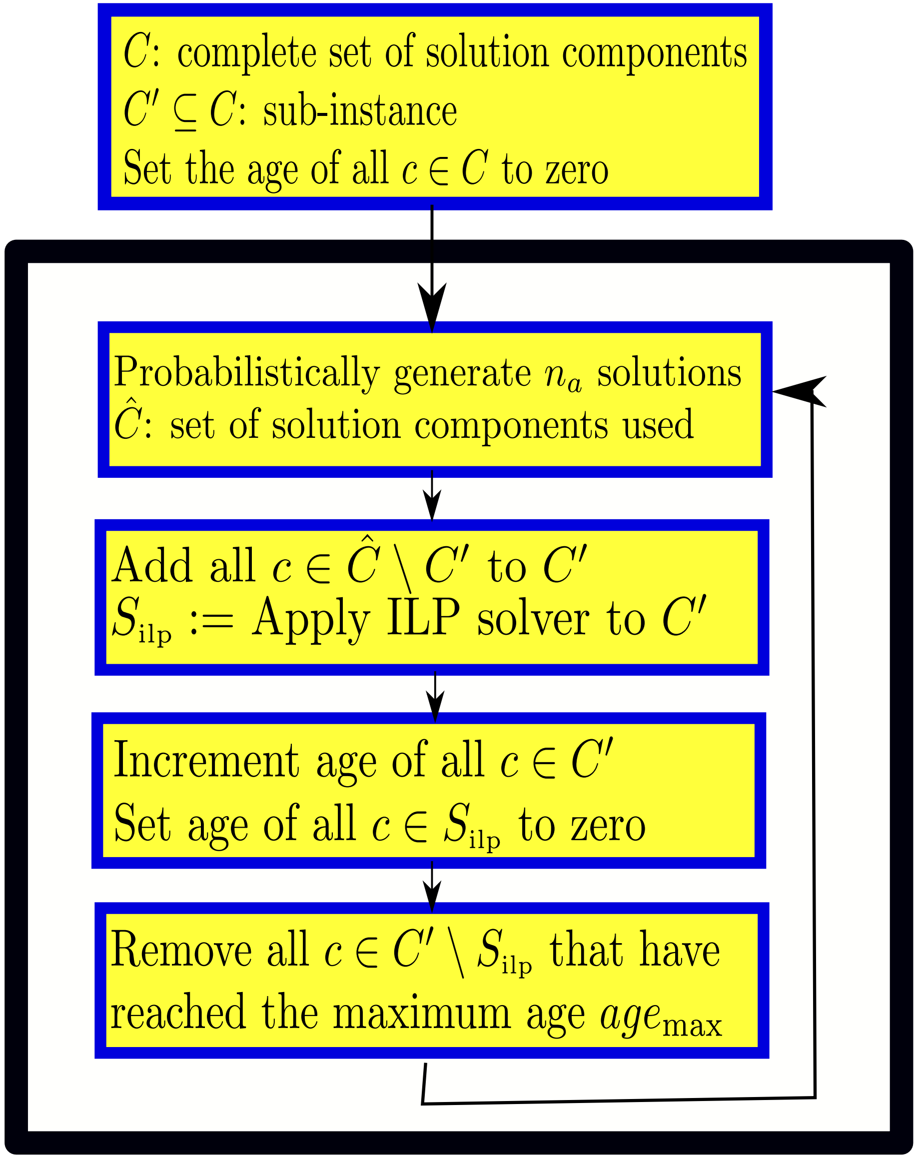
Observation: In the presence of a large number of solutions components, many of them only lead to bad solutions

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 (Cmsa)



Application example: Minimum Common String Partition (1)

Input:

1. Two related strings of length n over a finite alphabet Σ
2. **Note:** Two strings s_1 and s_2 are related iff the frequency of each letter in each string is equal.

Valid solutions:

- Generate a partition P_1 of non-overlapping substrings of s_1
- Generate a partition P_2 of non-overlapping substrings of s_2
- Solution $S = (P_1, P_2)$ is a valid solution iff $P_1 = P_2$
- Obj. function value: $f(S) := |P_1| = |P_2|$

Objective: Minimization

Minimum Common String Partition (2)

Example:

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

Related Literature

Basic facts:

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

Works from the literature:

- 2005: Greedy approach
- 2007: Introduction of approximation algorithms
- 2008: Study concerning fixed-parameter tractability (FPT)
- 2013: An ant colony optimization metaheuristic

Preliminaries

Definitions: Given input strings s_1 and s_2 ...

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

Common Block Example

Input strings: $s_1 = \textbf{AGACTG}$ and $s_2 = \textbf{ACTAGG}$ is as follows:

Set B of all common blocks:

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

Solution $\{\textbf{ACT}, \textbf{AG}, \textbf{G}\}$: $\mathcal{S} = \{b_1, b_2, b_{14}\}$

ILP Model (1)

$\boxed{\text{Input strings:}}$ $s_1 = \textbf{AGACTG}$ and $s_2 = \textbf{ACTAGG}$

$$B = \left\{ \begin{array}{l} b_1 = (\textbf{ACT}, 3, 1) \\ b_2 = (\textbf{AG}, 1, 4) \\ b_3 = (\textbf{AC}, 3, 1) \\ b_4 = (\textbf{CT}, 4, 2) \\ b_5 = (\textbf{A}, 1, 1) \\ b_6 = (\textbf{A}, 1, 4) \\ b_7 = (\textbf{A}, 3, 1) \\ b_8 = (\textbf{A}, 3, 4) \\ b_9 = (\textbf{C}, 4, 2) \\ b_{10} = (\textbf{T}, 5, 3) \\ b_{11} = (\textbf{G}, 2, 5) \\ b_{12} = (\textbf{G}, 2, 6) \\ b_{13} = (\textbf{G}, 6, 5) \\ b_{14} = (\textbf{G}, 6, 6) \end{array} \right\} \quad M1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

ILP Model (2)

$$\textbf{min} \sum_{i=1}^{m} x_i \tag{3}$$

subject to:

$$\sum_{i=1}^{m} |t_i| \cdot x_i = n \tag{4}$$

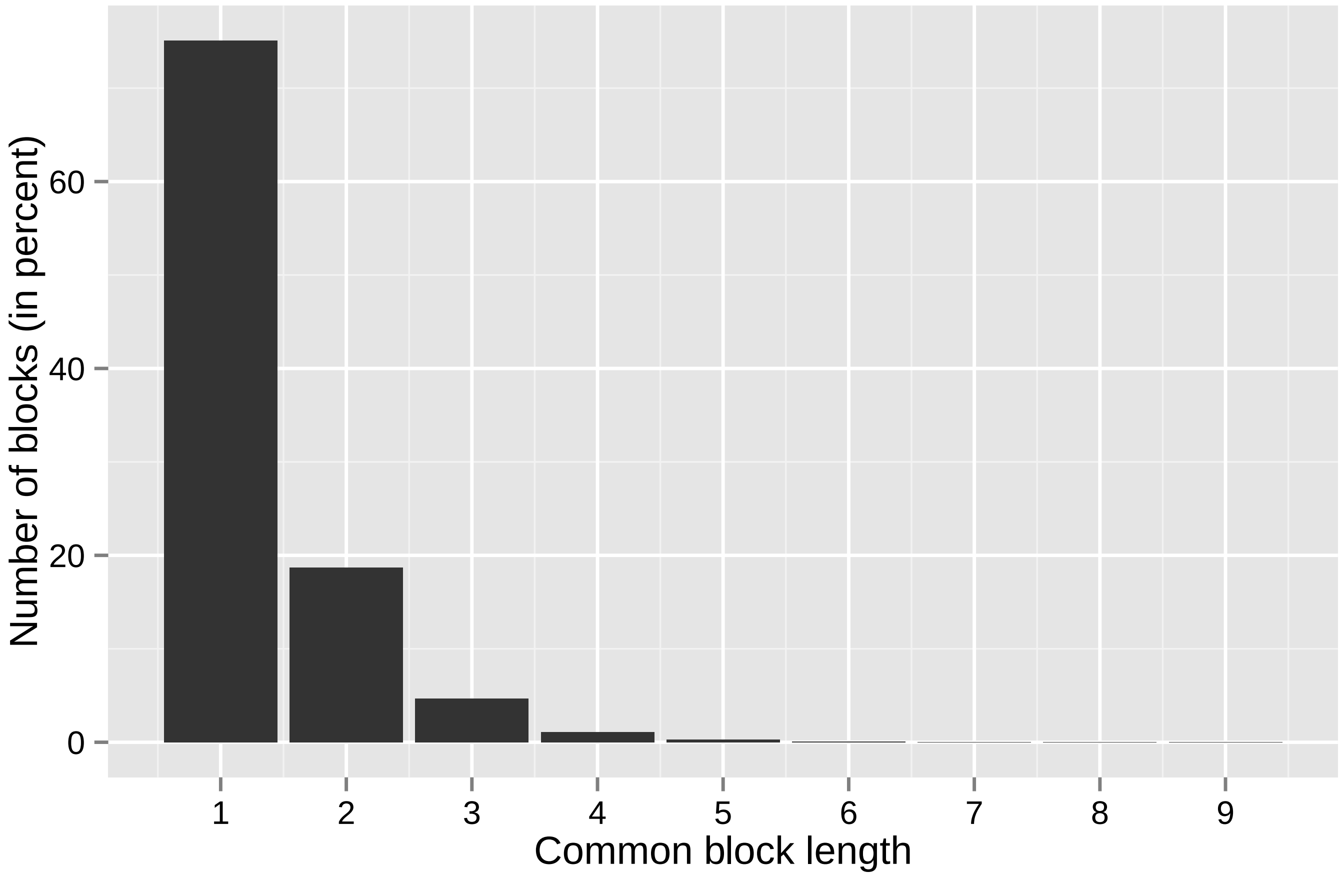
$$\sum_{i=1}^{m} M1_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{5}$$

$$\sum_{i=1}^{m} M2_{i,j} \cdot x_i = 1 \quad \text{for } j = 1, \ldots, n \tag{6}$$

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

Note: Very large number of solution components

Set of solution components: properties



Note: Most common blocks of length 1 and 2 will not appear in good solutions

Simple Greedy Algorithm

Given a valid partial solution S_{partial}: $B(S_{\text{partial}}) \subset B$ are the common blocks that may be used in order to extend S_{partial}

Pseudo-code:

1. $S_{\text{partial}} := \emptyset$
2. **while** S_{partial} is not a complete solution
   - Choose the longest common block b_i from $B(S_{\text{partial}})\}$
   - $S_{\text{partial}} := S_{\text{partial}} \cup \{b_i\}$

Note: This algorithm is used in CMSA in a probabilistic way

Benchmark instances and tuning

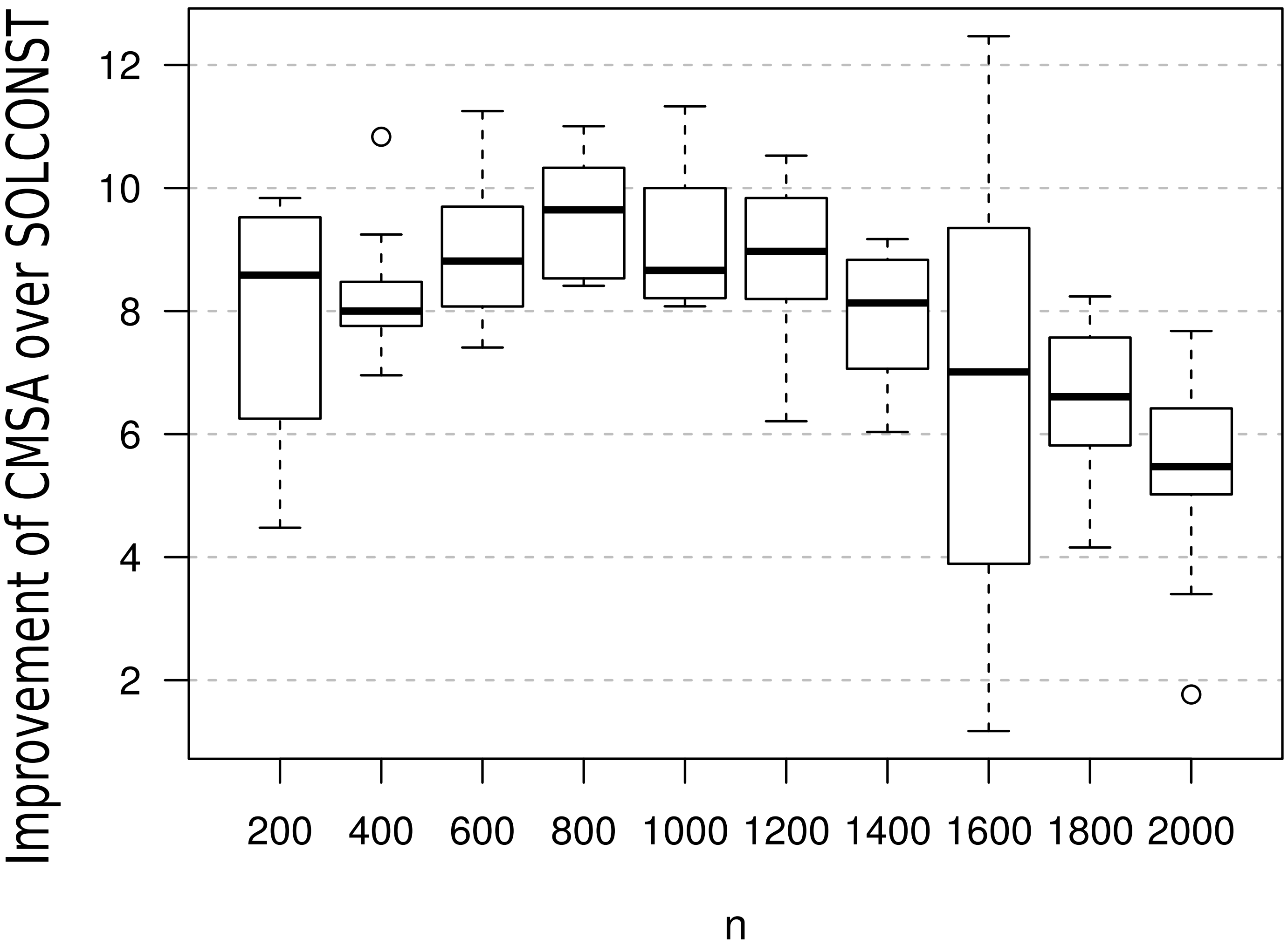
Benchmark instances: 300 instances

- String length $n \in \{200, 400, \ldots, 1800, 2000\}$
- Alphabet size $|\Sigma| \in \{4, 12, 20\}$

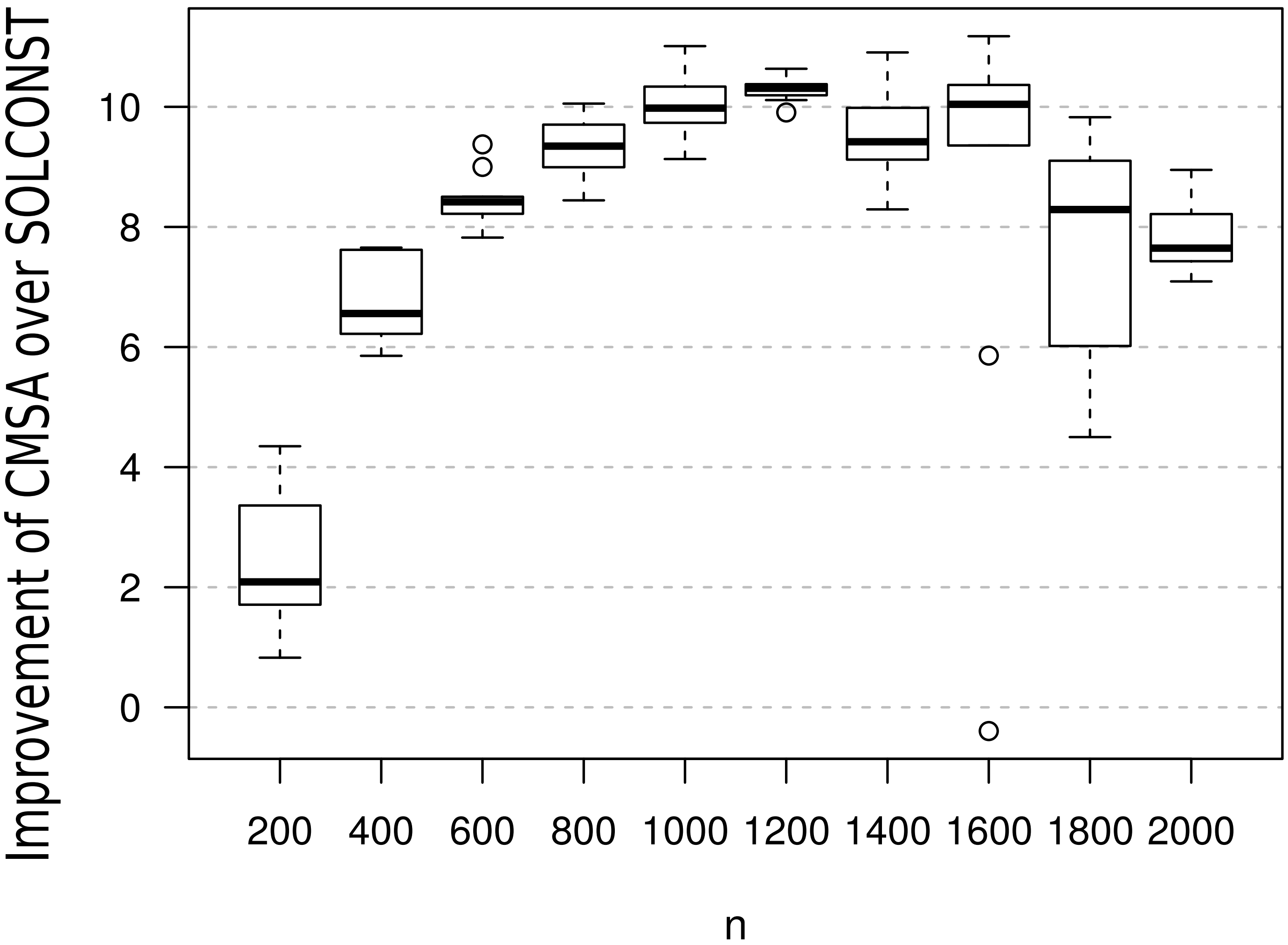
Tuning results with irace:

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

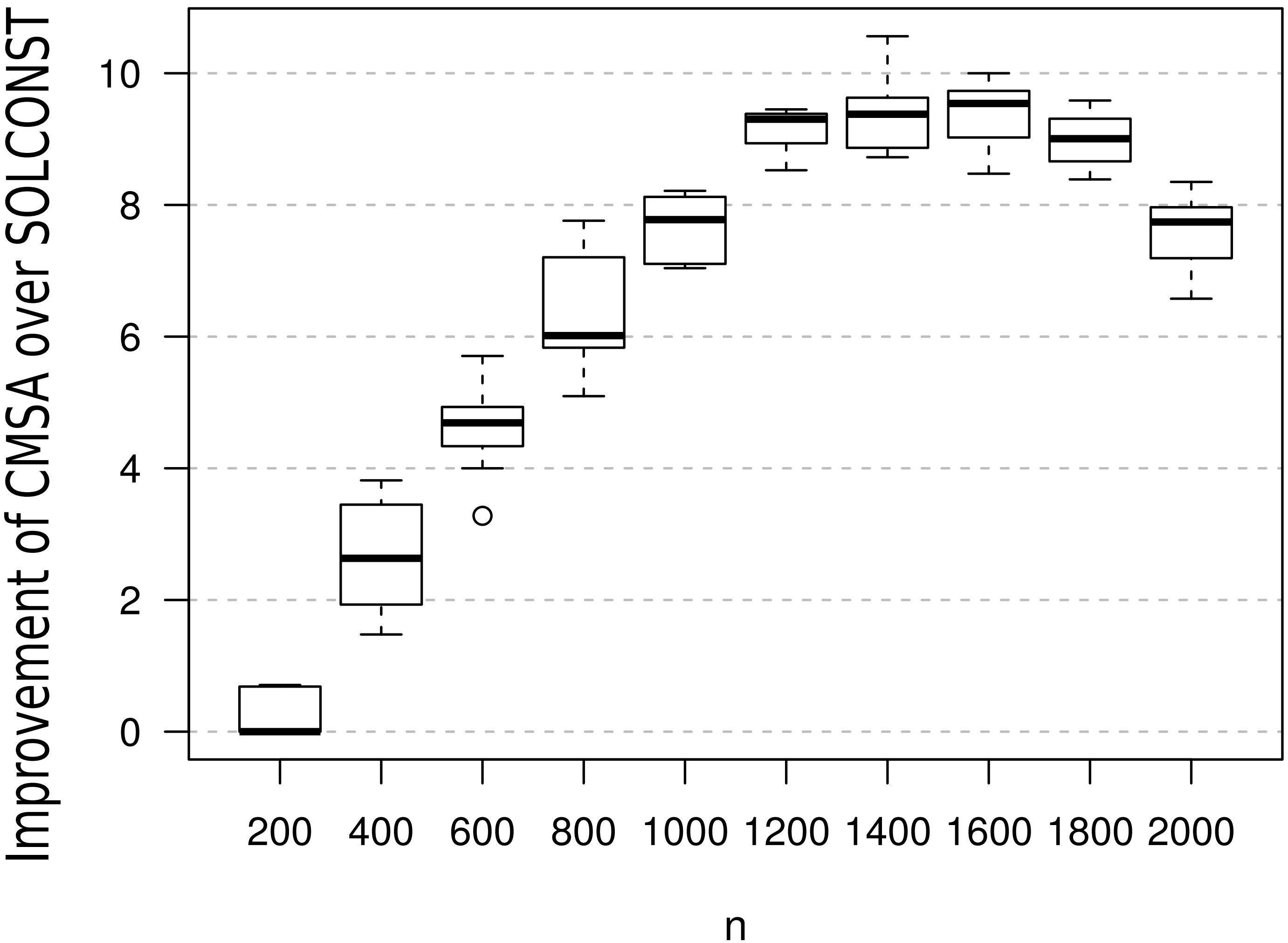
MCSP results: improvement over prob. GREEDY ($|\Sigma| = 4$)



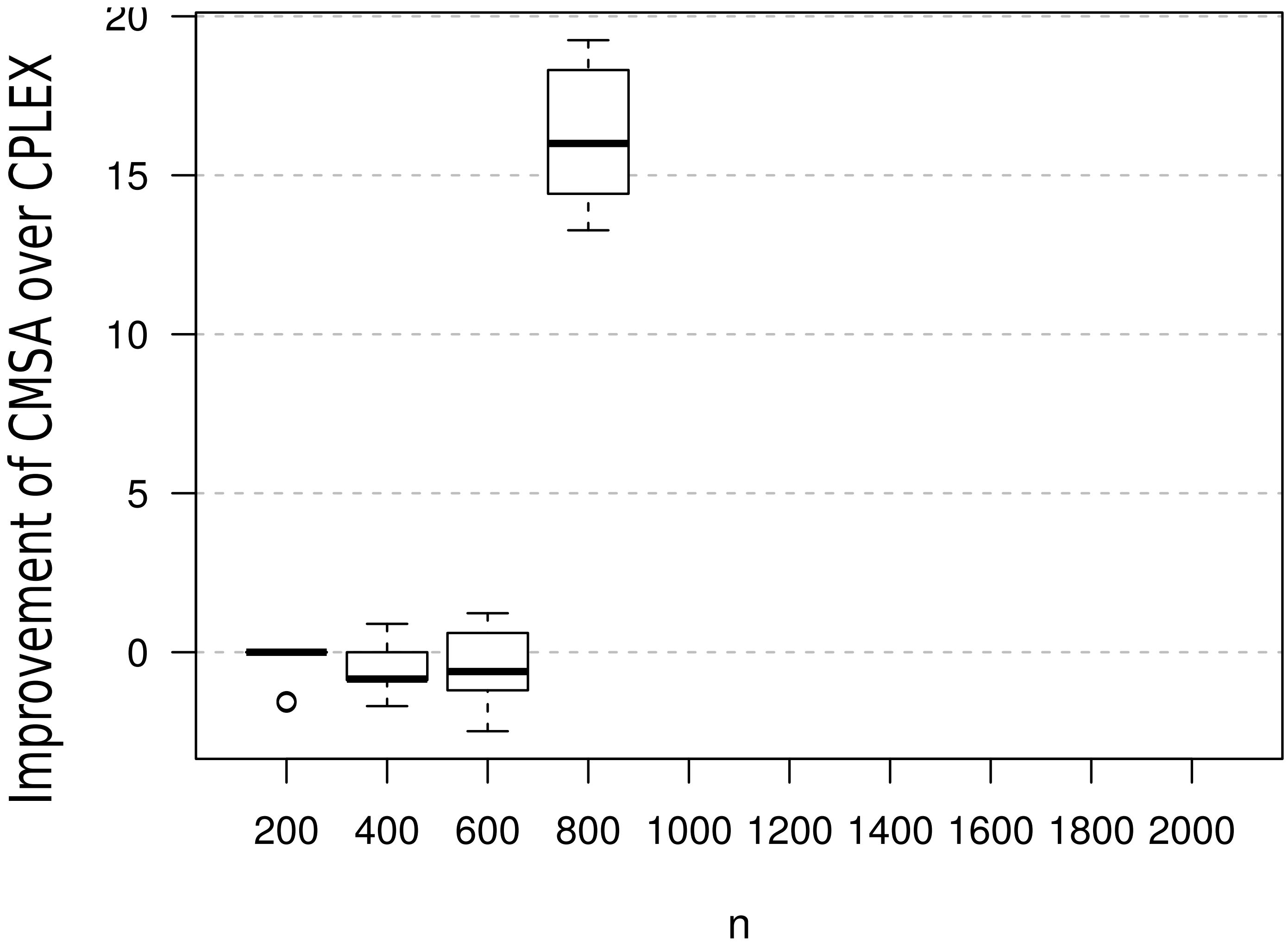
MCSP results: improvement over prob. GREEDY ($|\Sigma| = 12$)



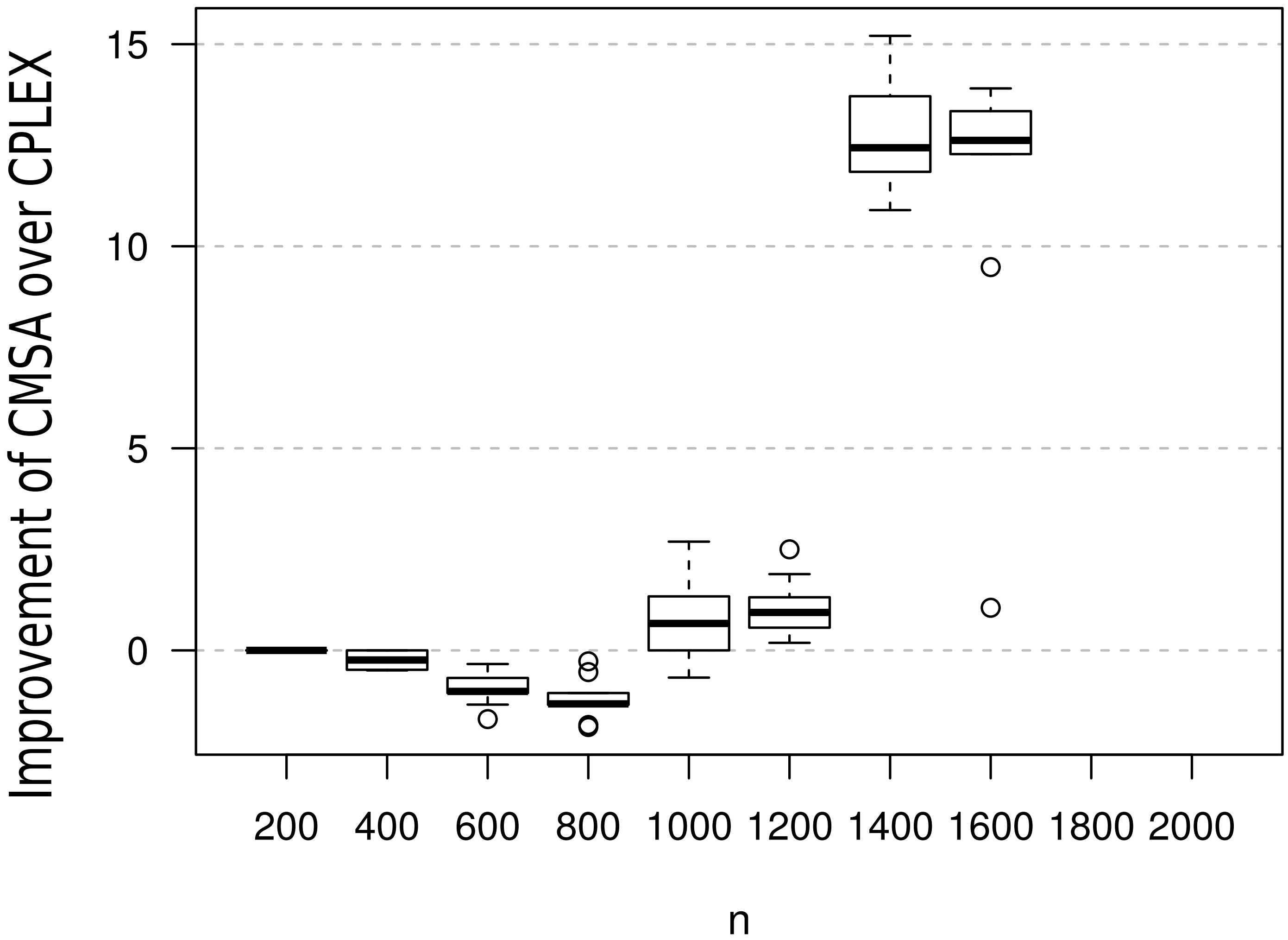
MCSP results: improvement over prob. GREEDY ($|\Sigma| = 20$)



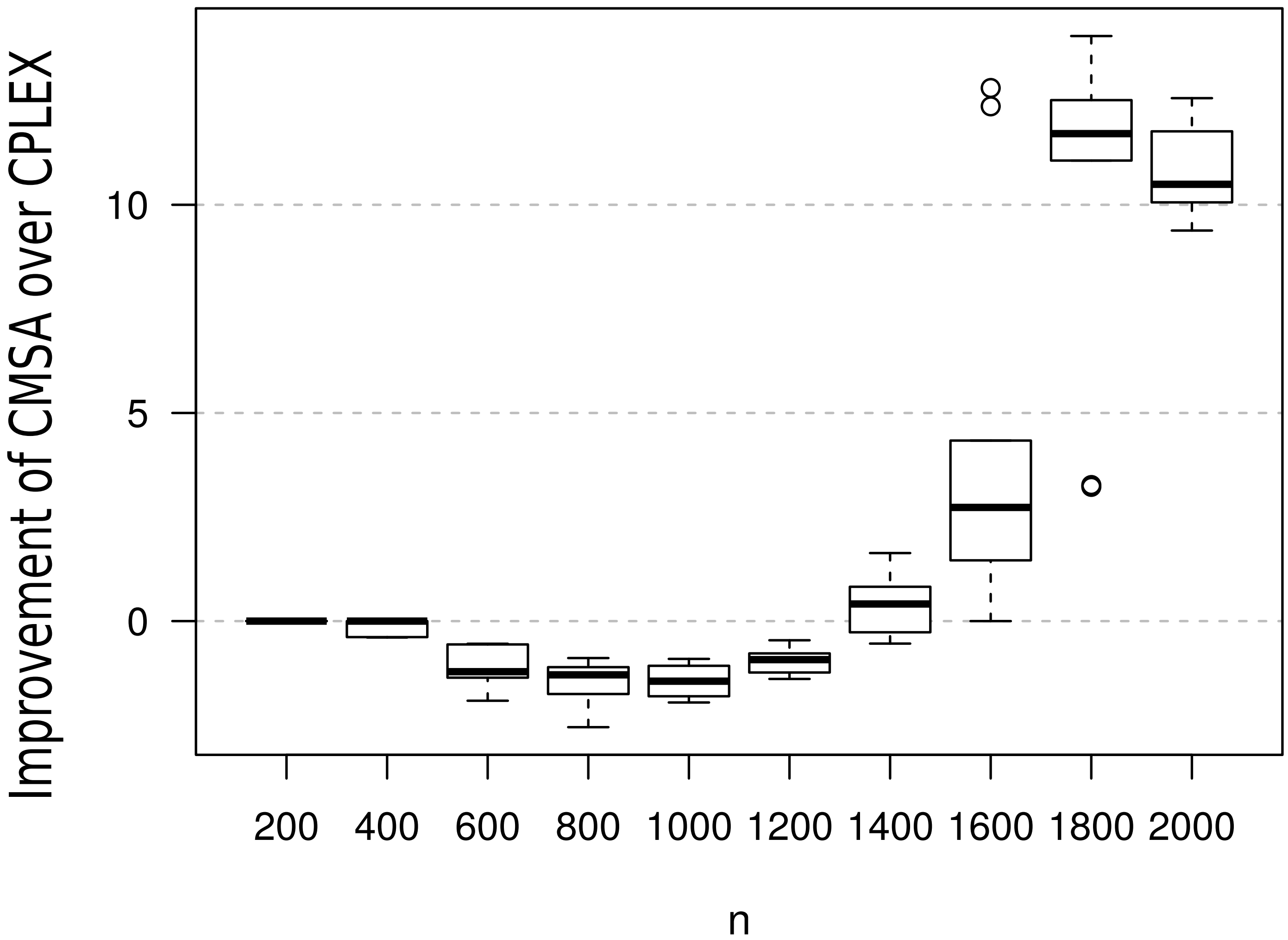
MCSP results: improvement over CPLEX ($|\Sigma| = 4$)



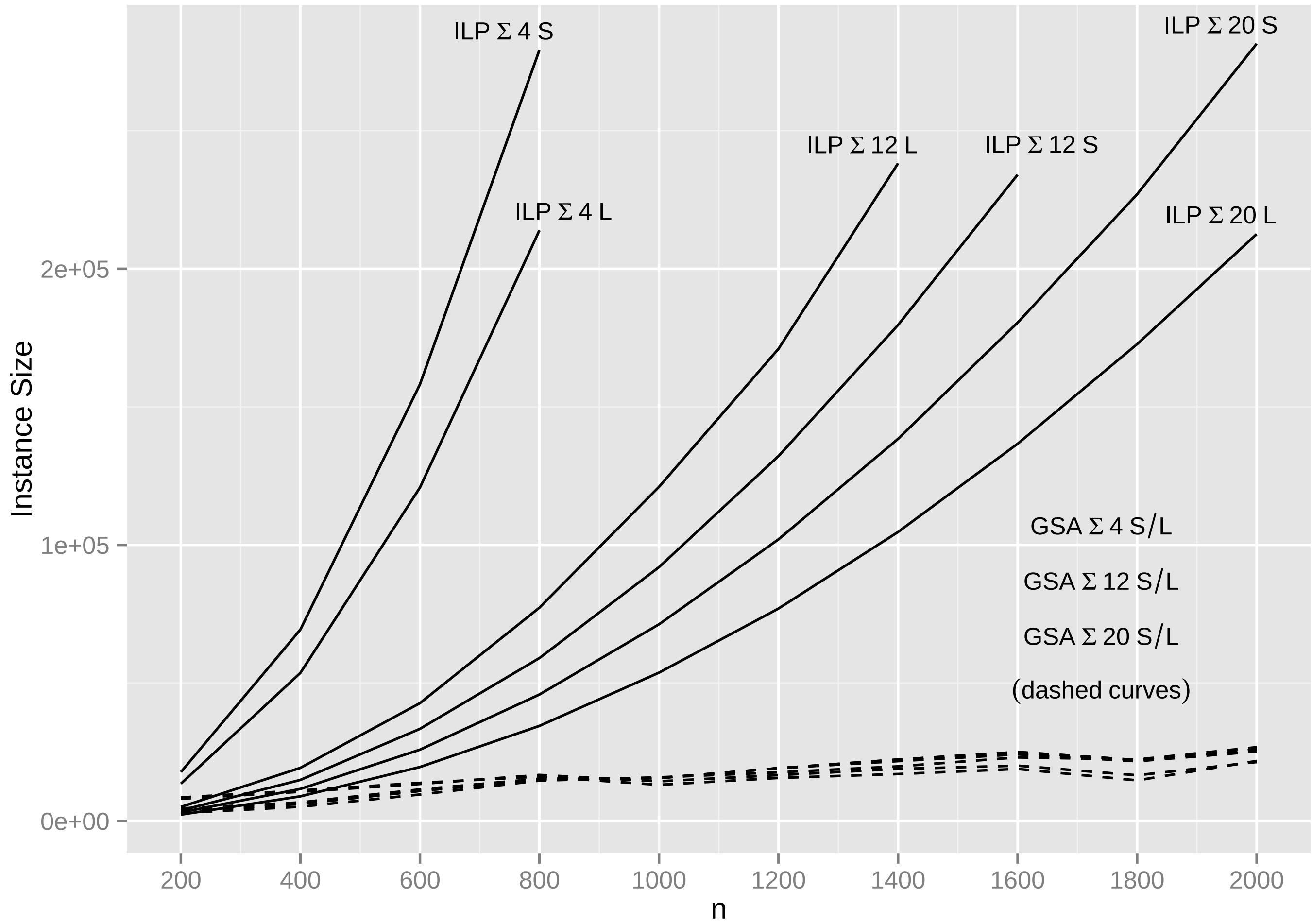
MCSP results: improvement over CPLEX ($|\Sigma| = 12$)



MCSP results: improvement over CPLEX ($|\Sigma| = 20$)

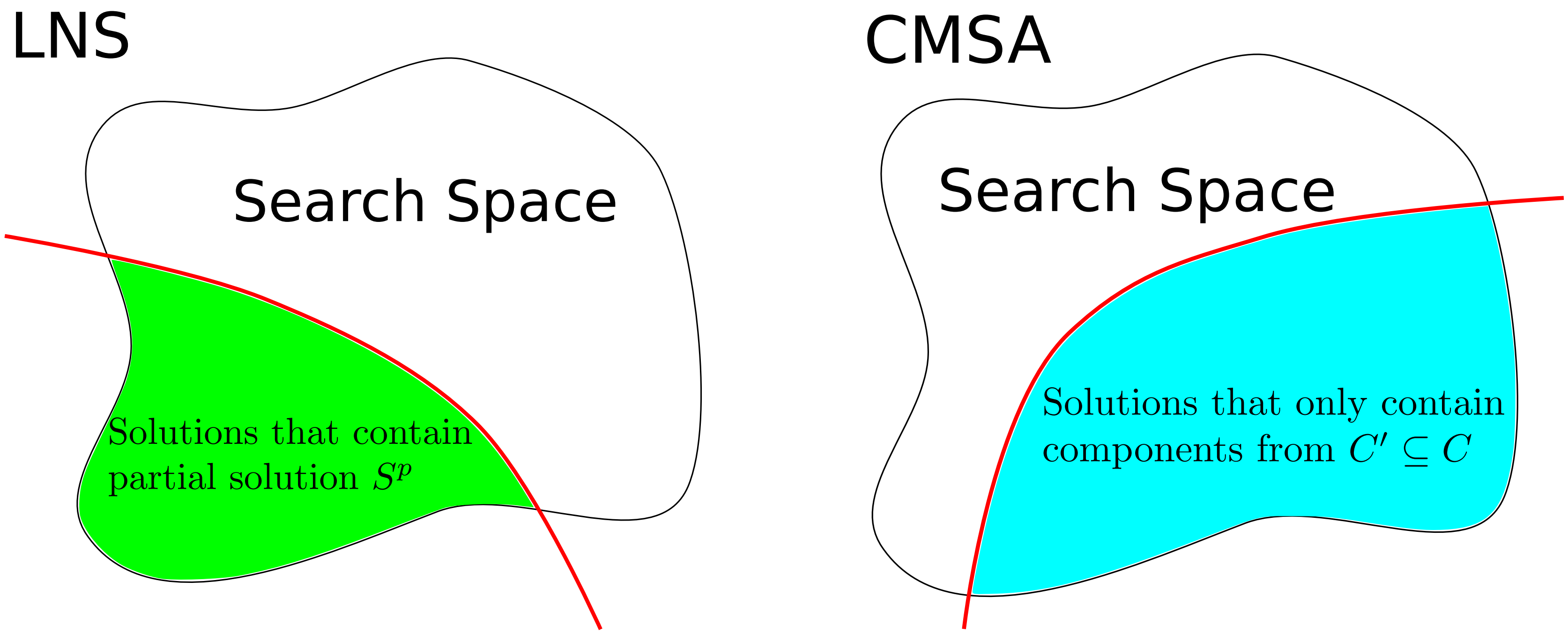


Evolution of the (sub-)instance sizes



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

Summary and Possible Research Directions

Summary:

- CMSA: A new hybrid algorithmm for combinatorial optimization
- Hypothesis:
  - LNS better for problems with a linear number solution components
  - CMSA better for problems with a super-linear number of components

Possible Research Directions:

- Solution construction: adaptive probabilities over time
- A more intelligent version of the aging mechanism
- Theoretical studies about the differences between LNS and CMSA

People involved in this research

Christian Blum

Borja Calvo

Pedro Pinacho

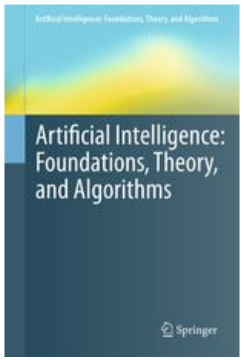
Jóse Antonio Lozano

Manuel López-Ibáñez

Questions?

Literature:

- C. Blum, B. Calvo. **A matheuristic for the minimum weight rooted arborescence problem**. *Journal of Heuristics*, 21(4): 479-499 (2015)
- C. Blum, J. Puchinger, G. R. Raidl, A. Roli. **Hybrid metaheuristics in combinatorial optimization: A survey**. *Applied Soft Computing*, 11(6): 4135–4151 (2011)



Forthcoming book: C. Blum, G. R. Raidl. Hybrid Metaheuristics – Powerful Tools for Optimization, Springer Series on Artificial Intelligence, 2015