

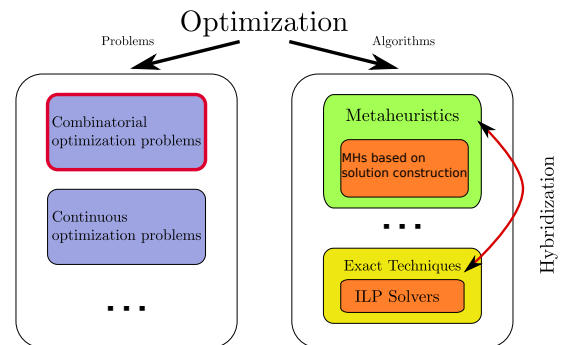
Hybrid Metaheuristics

Christian Blum

ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE (IIIA)
SPANISH NATIONAL RESEARCH COUNCIL (CSIC)



Preliminaries: preparing the grounds



How can we solve a combinatorial problem exactly?

Exact solution methods:

- ▶ 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 for Minimization (1)

General statements:

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

Clarification: on the board!

Generic Branch & Bound for Minimization (2)

```

1:  $B := \infty$ 
2: if heuristic available then  $s := \text{Apply a heuristic}$ ;  $B := f(s)$ ;  $s_{\text{bsf}} := s$ 
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  $s'$  and  $f(s') < B$  then
7:      $B := f(s')$  and  $s_{\text{bsf}} := s'$ 
8:   else
9:     Use the branching rule to produce a set of sub-problems  $N_{\text{sub}}$  from  $N$ 
10:    for all  $N_i \in N_{\text{sub}}$  do
11:      if  $g(N_i) \leq B$  then add  $N_i$  to  $Q$  end if
12:    end for
13:  end if
14: end while
  
```

Generic Branch & Bound for Minimization (3)

The queue Q : different ways of handling

- ▶ **Last-in-first-out** (LIFO): results in a **depth-first** algorithm
- ▶ **First-in-first-out** (FIFO): results in a **breadth-first** algorithm
- ▶ **Priority queue depending on $g()$** : results in a **best-first** algorithm

Note: when no heuristic is available, the FIFO variant is preferable.

Dynamic Programming

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 sub-problem space:** Should be of moderate size (polynomial)

Example: Longest Common Subsequence (LCS) Problem

Definitions:

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

Observations:

1. **Case 1:** $x[n] = y[m]$ $\text{LCS}(x, y) = \text{LCS}(x_{n-1}, y_{m-1}) + x[n]$

Example: Longest Common Subsequence (LCS) Problem

Observations (continued):

1. **Case 2:** $x[n] \neq y[m]$
 - **Sub-case 2.1:** $\text{LCS}(x, y)$ ends with $x[n]$. **Then:**
 $\text{LCS}(x, y) = \text{LCS}(x, y_{m-1})$
 - **Sub-case 2.2:** $\text{LCS}(x, y)$ ends with $y[m]$. **Then:**
 $\text{LCS}(x, y) = \text{LCS}(x_{n-1}, y_m)$

Resulting DP formula: for any $i \in \{0, 1, \dots, n\}$ and $j \in \{1, \dots, m\}$

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

Example: Longest Common Subsequence (LCS) Problem

Example:

| | | 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)

What is it? Computational systems based on constraints

How does it work?

- **Phase 1:**
 - ★ Express CO problem in terms of a discrete problem (variables+domains)
 - ★ Define ("post") constraints among the variables
 - ★ The **constraint solver** reduces the variable domains
 - ... before solution construction
 - ... during solution construction
- **Phase 2:** Labelling
 - ★ Search through the remaining search tree
 - ★ Possibly "post" additional constraints

Example of Domain Reduction (1)

Simple example: minimize $f(X, Y, Z) \mapsto \mathbf{R}$

subject to

$$\begin{aligned} X &\in \{1, \dots, 8\} \\ Y, Z &\in \{1, \dots, 10\} \\ X &\neq 7, Z \neq 2 \\ X - Z &= 3Y \end{aligned}$$

Constraint propagation:

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

Example of Domain Reduction (2)

- ▶ **Step 2:** Use $X - Z = 3Y$
 1. Because of the domains of X and Z : $X - Z < 8$
 2. $\Rightarrow 3Y < 8$
 3. $\Rightarrow Y \leq 2$
 4. $\Rightarrow Y \in \{1, 2\}$
- ▶ **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. $\Rightarrow X \in \{4, 5, 6, 8\}$ and $Z \in \{1, 3, 4, 5\}$

Topic Outline

- ▶ **Hybrid Metaheuristics (HMs)**
 - ★ Definition
 - ★ Classification of HMs
- ▶ **Examples**
 - ★ Metaheuristics with **Metaheuristics** (ILS)
 - ★ Metaheuristics with **Constraint Programming** (ACO)
 - ★ Metaheuristics with **Tree Search** (VNS)
 - ★ Metaheuristics with **Problem Relaxation** (TS)
 - ★ Metaheuristics with **Dynamic Programming** (EC)
- ▶ **Examples from my own recent work**

Hybrid Metaheuristics

A short introduction

Metaheuristics: Time Line 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 Randomized Adaptive Search Procedure (GRASP) [Resende, 1995]
- ▶ Particle Swarm Optimization (PSO) [Eberhart, Kennedy, 1995]
- ▶ Guided Local Search (GLS) [Voudouris, 1997]
- ▶ Iterated Local Search (ILS) [Stützle, 1999]
- ▶ Variable Neighborhood Search (VNS) [Mladenović, 1999]

Note: No essentially new idea since ≈ 2000

Hybrid metaheuristics (1)

Definition: What is a **hybrid** metaheuristic?

- ▶ **Problem:** can not be very well defined

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
- ▶ ILP techniques

Hybrid metaheuristics (2)

Note: Lack of a precise definition is often **subject to criticism**

History:

- ▶ For a long time the different **communities** co-existed quite **isolated**
- ▶ **Hybrid approaches** were developed already early, but only **sporadically**
- ▶ Only **since about 10 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

Hybrid metaheuristics: classification (1)

References for the **classification** of hybrid metaheuristics

- ▶ 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

Hybrid metaheuristics: classification (2)

What is hybridized? Metaheuristics with ...

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

Hybrid metaheuristics: classification (3)

What is the level of 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

Hybrid metaheuristics: classification (4)

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-scale neighborhood search
 4. Using metaheuristics for finding good upper bounds in branch & bound

Hybrid metaheuristics: classification (5)

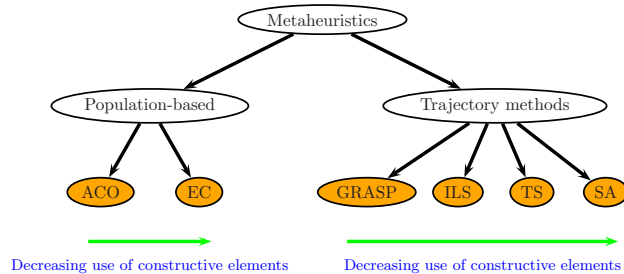
What is the order of execution?

- ▶ **Sequential**: results of earlier executed algorithms are used in later algorithms
- ▶ **Interleaved**. For example:
 1. Using metaheuristics for node selection in branch & bound
 2. Exact algorithms as decoders in decoder-based approaches
- ▶ **Parallel**
 1. **Granularity**: fine-grained versus coarse-grained
 2. **Hardware**: homogeneous versus heterogeneous
 3. etc.

Representative Examples

- ▶ **Metaheuristics with Metaheuristics**
- ▶ Metaheuristics with Constraint Programming
- ▶ Metaheuristics with Mathematical Programming
- ▶ Metaheuristics with Problem Relaxation
- ▶ Metaheuristics with Dynamic Programming

Characteristics of Different Metaheuristics



- ▶ **Advantage of pop.-based methods:** Diversification ability
- ▶ **Advantage of trajectory methods:** Intensification ability

What does that mean 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**)

Other examples:

- ▶ **Population-based iterated local search**
- ▶ **Multi-level techniques**

Iterated Local Search

Pseudo-code:

```

1:  $s \leftarrow \text{GenerateInitialSolution}()$ 
2:  $s \leftarrow \text{LocalSearch}(s)$ 
3: while termination conditions not met do
4:    $s' \leftarrow \text{Perturbation}(s, \text{history})$ 
5:    $\hat{s}' \leftarrow \text{LocalSearch}(s')$ 
6:    $s \leftarrow \text{ApplyAcceptanceCriterion}(\hat{s}', s, \text{history})$ 
7: end while
  
```

Key components:

- ▶ Perturbation mechanism
- ▶ Local search

Population-Based Iterated Local Search

```

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

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

Population-Based ILS: QAP Example

Algorithm components: (remember: solutions are permutations)

- ▶ **Neighborhood for local search:** 2-opt, first-improvement
- ▶ **Perturbation:** random k -opt move
- ▶ **Additional feature:** dynamic setting of k
 1. $k \in [k_{\min}, k_{\max}]$
 2. Before first iteration: $k := k_{\max}$
 3. **Then:** decrease k at each iteration until $k = k_{\min}$
 4. **Then:** in case the new solution is accepted: $k := k_{\min}$
 5. **Otherwise:** $k := k + 1$

The Multi-level Framework (1)

General references:

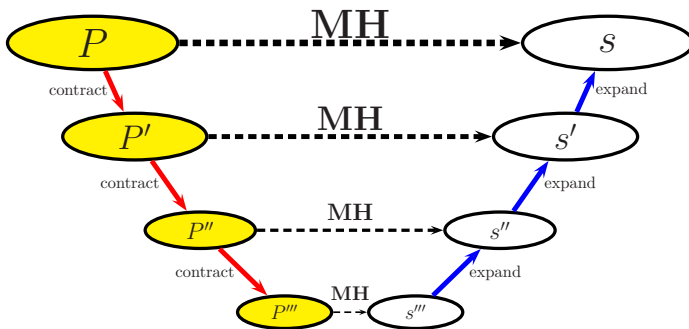
- ▶ C. Walshaw. **Multilevel refinement for combinatorial optimisation**, *Annals of Operations Research*, 131:325–372, 2004
- ▶ C. Walshaw. **Multilevel refinement for combinatorial optimisation: boosting metaheuristic performance**, In: *Hybrid Metaheuristics—An Emerging Approach to Optimization*, 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

The Multi-level Framework (2)

The multi-level framework:



The Multi-level Framework (3)

Specific references:

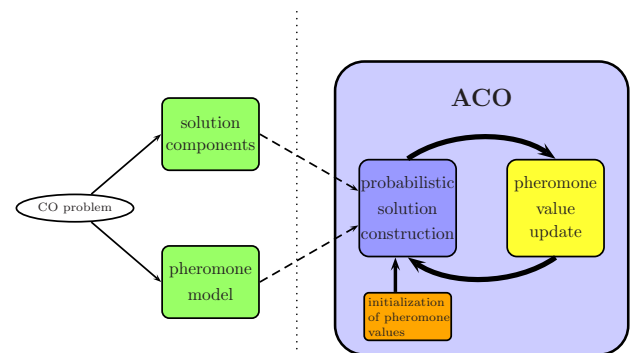
- ▶ C. Walshaw. A multilevel approach to the travelling salesman problem, *Operations Research*, 50(5):862–877, 2002
- ▶ T. G. Crainic, Y. Li, and M. Toulouse. A first multilevel cooperative algorithm for capacitated multicommodity network design, *Computers & Operations Research*, 33(9):2602–2622, 2006
- ▶ 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

Note: Example for the TSP on the board!

Representative Examples

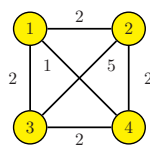
- ▶ Metaheuristics with Metaheuristics
- ▶ **Metaheuristics with Constraint Programming**
- ▶ Metaheuristics with Mathematical Programming
- ▶ Metaheuristics with Problem Relaxation
- ▶ Metaheuristics with Dynamic Programming

Ant Colony Optimization (ACO)



ACO+CP: TSP Example

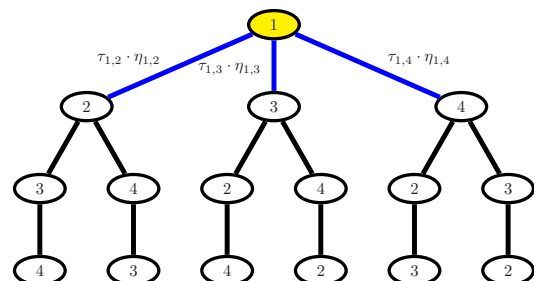
Example instance:



Solution construction: Constructive mechanism of the nearest-neighbor heuristic (starting from city 1)

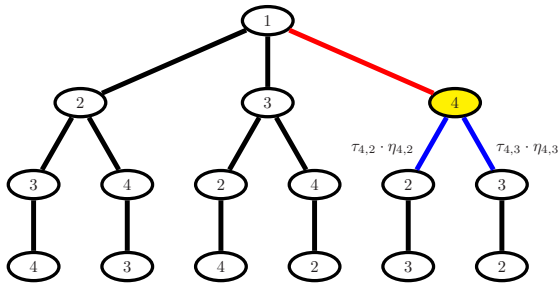
ACO as a tree search algorithm

1st construction step:



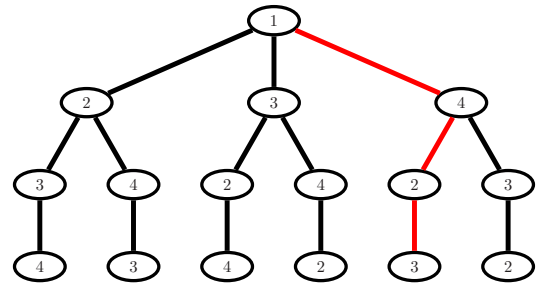
ACO as a tree search algorithm

2nd construction step:



ACO as a tree search algorithm

3rd construction step:



ACO hybridized with constraint programming (1)

References:

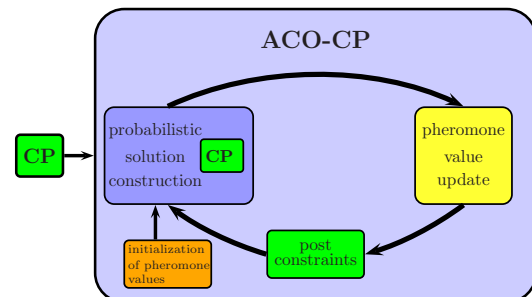
- ▶ B. Meyer and A. Ernst. **Integrating ACO and Constraint Propagation**, In: *Proceedings of ANTS 2004*, volume 3172 of Springer LNCS, pages 166–177, 2004
- ▶ M. Khichane, P. Albert, and C. Solnon. **CP with ACO**, In: *Proceedings of CPAIOR 2008*, volume 5015 of Springer LNCS, pages 328–332, 2008

General idea:

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

ACO hybridized with constraint programming (5)

ACO-CP hybrid:



ACO hybridized with constraint programming (6)

Evaluation:

- ▶ **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

ACO-CP: Good with intermediate number of feasible solutions

Problem:

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

Representative Examples

- ▶ Metaheuristics with Metaheuristics
- ▶ Metaheuristics with Constraint Programming
- ▶ **Metaheuristics with Mathematical Programming**
- ▶ Metaheuristics with Problem Relaxation
- ▶ Metaheuristics with Dynamic Programming

Large neighborhood search (1)

General references:

- ▶ R. K. Ahuja, O. Ergun, J. B. Orlin, and A. P. Punnen. **A survey of very large-scale neighborhood search techniques**, *Discrete Applied Mathematics*, 123(1-3):75–102, 2002
- ▶ M. Chiarandini, I. Dumitrescu, and T. Stützle. **Very Large-Scale Neighborhood Search: Overview and Case Studies on Coloring Problems**, In: *Hybrid Metaheuristics—An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*, pages 117–150, Springer Verlag, Berlin, Germany, 2008

Key issues in local search:

- ▶ Defining an appropriate neighborhood structure
- ▶ Choosing a way of examining the neighborhood of a solution

Large neighborhood search (2)

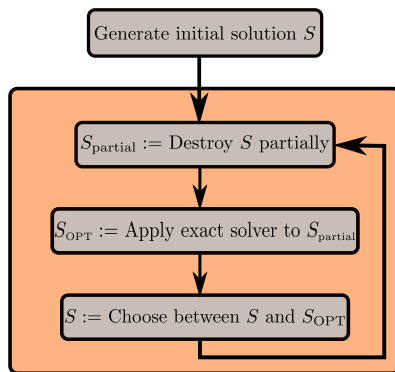
General tradeoff:

- ▶ **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 of examining large neighborhoods:

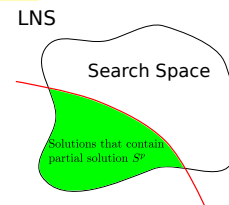
- ▶ Heuristically
- ▶ In some cases an **efficient exact technique** may exist

Generic Large Neighborhood Search



Crucial Aspect of Large Neighborhood Search

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

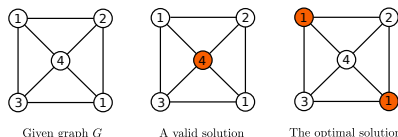


Example: Minimum weight dominating set

- ▶ **Given:** un-directed 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, iff

$$\forall v_i \in V: N[v_i] \cap S \neq \emptyset$$
- ▶ **Optimization objective:** Finding a solution S^* that minimizes

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



ILP Model

$$\begin{aligned} \min \quad & \sum_{v_i \in V} w(v_i) \cdot x_i \\ \text{subject to:} \quad & \sum_{v_j \in N[v_i]} x_j \geq 1 \quad \text{for } v_i \in V \\ & x_i \in \{0, 1\} \quad \text{for } v_i \in V \end{aligned} \tag{1}$$

$$\tag{2}$$

Note:

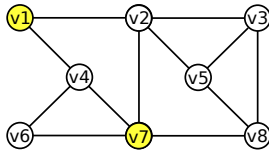
- ▶ In this ILP model: **number of variables and constraints is linear**
- ▶ How to find the best solution containing S_{partial} :
Add these constraints to the model: $x_i = 1 \quad \text{para } v_i \in S_{\text{partial}}$

Generating the initial solution: GREEDY (1)

Definitions:

- V_{cov} : set of **covered nodes** (w.r.t. a partial solution)
- $d(v|V_{\text{cov}})$: **current degree** of a node v (without 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:

```

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

```

Partial Destruction of a Solution

Standard method: remove a certain **percentage $\text{perc}_{\text{dest}}$** of all nodes in S_{cur}

How to choose these nodes:

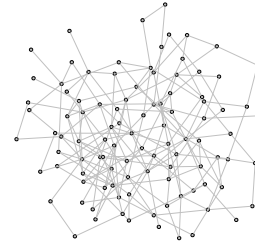
- Destruction $\text{type}_{\text{dest}} = \text{random}$: random selection
- Destruction $\text{type}_{\text{dest}} = \text{biased}$: selection biased by a greedy function

Choosing a value for $\text{perc}_{\text{dest}}$:

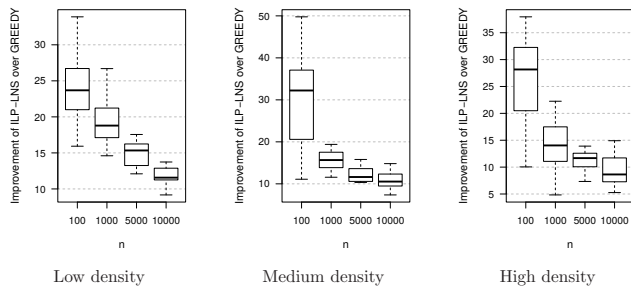
- Dynamic choice from $[\text{perc}_{\text{dest}}^d, \text{perc}_{\text{dest}}^u]$
- Initially $\text{perc}_{\text{dest}} := \text{perc}_{\text{dest}}^d$
- When no improving solution is found: $\text{perc}_{\text{dest}} := \text{perc}_{\text{dest}} + 5$
- When an improving solution is found: $\text{perc}_{\text{dest}} := \text{perc}_{\text{dest}}^d$

Instances

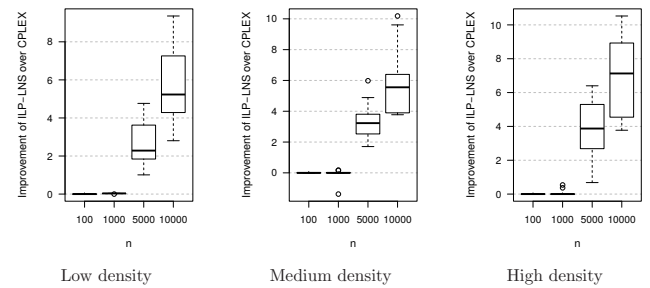
- Random graphs with $|V| = \{100, 1000, 5000, 10000\}$ nodes
- Different edge probabilities e_p (low, medium, high):
 - ★ For $|V| = 100$: $e_p \in \{0.03, 0.04, 0.05\}$
 - ★ For $|V| > 100$: $e_p \in \{0.01, 0.03, 0.05\}$



Results: Improvement over GREEDY (en %)



Results: Improvement over CPLEX (en %)



Second LNS Example: Biological Problem 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

The Founder Sequence Reconstruction Problem (FSRP)

- **Given:** A set of m **recombinants** $\mathcal{C} = \{C_1, \dots, C_m\}$
 - ★ **Here:** $\forall i, C_i$ is a binary string of length n
- **Candidate solution:** A set of k **founders** $\mathcal{F} = \{F_1, \dots, F_k\}$
 - ★ **Here:** $\forall j, F_j$ is a binary string of length n
- A **solution is valid** 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 $Fr_{i1}Fr_{i2}\dots Fr_{ip_i}$, such that each fragment Fr_{ij} appears at the same position in at least one of the founders
- **Given \mathcal{F}** , a **minimal decomposition**, where the number of **breakpoints** $\sum_{i=1}^n p_i - n$ is minimal, can be derived in polynomial time

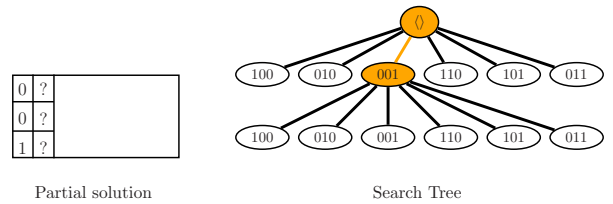
FSRP: Optimization Goal, and Example

- **Optimization goal:** Given k , find a valid solution \mathcal{F}^* that minimizes $f(\cdot)$

Example:

| | | |
|----------------------------|------------------------|-------------------|
| 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 |
| Recombinants \mathcal{C} | Founders \mathcal{F} | Reconstruction |

Branch & Bound Algorithm: RECBLOCK



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

Branch & Bound Algorithm: RECBLOCK

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}

| | |
|---|--|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |

Variable Neighborhood Descent (VND)

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

- 1: INPUT: a solution s , r_{\max} neighborhood functions
- 2: $r := 1$
- 3: **while** $r \leq r_{\max}$ **do**
- 4: $s' := \text{PickBestNeighbor}(s, \mathcal{N}_r)$
- 5: **if** $f(s') < f(s)$ **then**
- 6: $s := s'$
- 7: $r := 1$
- 8: **else**
- 9: $r := r + 1$
- 10: **end if**
- 11: **end while**
- 12: OUTPUT: a (possibly) improved solution s

Hybrid VND for the FSRP

```

1: INPUT: a solution  $s$ , number  $k$  of founders
2:  $r := 1$ 
3: while  $r \leq k$  do
4:    $\hat{s} := \text{DeleteFounders}(s, r)$ 
5:    $s' := \text{RECBLOCK}(\hat{s})$ 
6:   if  $f(s') < f(s)$  then
7:      $s := s'$ 
8:      $r := 1$ 
9:   else
10:    if maximal number of trials reached then  $r := r + 1$ 
11:  end if
12: end while
13: OUTPUT: a (possibly) improved solution  $s$ 

```

Representative Examples

- ▶ Metaheuristics with Metaheuristics
- ▶ Metaheuristics with Constraint Programming
- ▶ Metaheuristics with Mathematical Programming
- ▶ **Metaheuristics with Problem Relaxation**
- ▶ Metaheuristics with Dynamic Programming

Problem Relaxation

Observe: Problem relaxations can be obtained (among others) by

- ▶ Simplifying constraints of an IP formulation
- ▶ Dropping constraints of an IP formulation (e.g. integrality constraints)
- ▶ Moving constraints in terms of penalties to the objective function (e.g. Lagrangian relaxation)

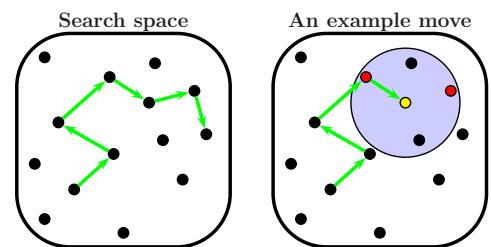
Use of relaxations:

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

Tabu Search

Main feature: Use of tabu lists for storing solution features

Note: Tabu lists are used to avoid going back to already visited solutions



Hybrid Tabu Search

Specific Reference:

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

Characteristics:

- ▶ Collaborative hybridization approach
- ▶ 1st algorithm phase: problem relaxation is used to produce a bunch of promising solutions
- ▶ 2nd algorithm phase: tabu search is used to search around these solutions

The 0-1 Multidimensional Knapsack Problem (MKP)

Given:

- ▶ n objects, each object i with a profit c_i
- ▶ m resources, each resource j with a capacity b_j
- ▶ Each object i has a requirement a_{ij} of each resource j

IP formulation:

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

subject to

$$\begin{aligned} a_{ij} \cdot x_i &\leq b_j & j &= 1, \dots, m \\ x_i &\in \{0, 1\} & i &= 1, \dots, n \end{aligned}$$

First Algorithm Phase

Main ideas:

- ▶ **Dropping** the integrality constraints
- ▶ For all k such that $0 \leq k_{\min} \leq k \leq k_{\max} \leq n$ solve

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

subject to

$$\begin{aligned} a_{ij} \cdot x_i &\leq b_j & j &= 1, \dots, m \\ 0 \leq x_i &\leq 1 & i &= 1, \dots, n \\ \sum_{i=1}^n x_i &= k \end{aligned}$$

Second Algorithm Phase

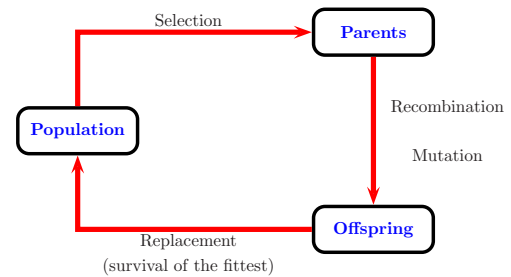
Main ideas:

- ▶ **From the 1st phase solutions:** Produce integer solutions by rounding
- ▶ Use tabu search to search in the vicinity of these integer solutions
- ▶ **Definition of vicinity:** maximum Hamming distance

Representative Examples

- ▶ Metaheuristics with Metaheuristics
- ▶ Metaheuristics with Constraint Programming
- ▶ Metaheuristics with Mathematical Programming
- ▶ Metaheuristics with Problem Relaxation
- ▶ **Metaheuristics with Dynamic Programming**

Evolutionary Algorithms (EAs)



The k -Cardinality Tree (KCT) Problem (1)

Specific 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

Definition: The k -cardinality tree problem

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 cardinality $k < |V|$

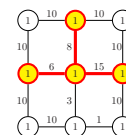
The k -Cardinality Tree (KCT) Problem (2)

Let \mathcal{T}_k be the set of all trees in G with exactly k edges

Optimization goal: Find a k -cardinality tree $T_k \in \mathcal{T}_k$ which minimizes

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

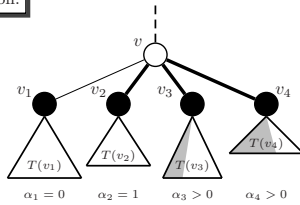
Example: A 3-cardinality tree



Dynamic Programming for the KCT Problem

Observation: KCT can be solved optimally if G is a tree

Graphical explanation:



Complexity: $\mathcal{O}(k^2|V|)$

Utilizing DP for Crossover

Given: Two k -trees T_1 and T_2 (**parents**)

Case 1: T_1 and T_2 have a least one node in common

1. **Merge** T_1 and $T_2 \leftrightarrow$ A graph G_c
2. Generate a **minimum spanning tree** T of G_c
3. **Use DP** for obtaining the best k -tree in T

Case 2: T_1 and T_2 do not have any node in common

1. **Use tree construction** to increase T_1 until it **touches** $T_2 \leftrightarrow T$
2. **Use DP** for obtaining the best k -tree in T

Examples From My Recent Work

Construct, Merge, Solve & Adapt (CMSA)

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 fulfilled?

Construct, Merge, Solve & Adapt: Principal Idea

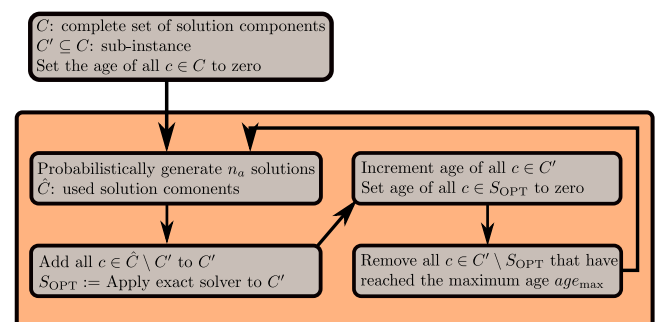
Observation: In the presence of a large number of solutions components, many of them only lead to bad solutions

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

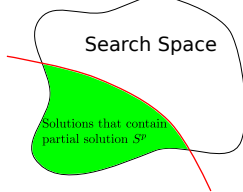
Construct, Merge, Solve & Adapt: Flow Diagram



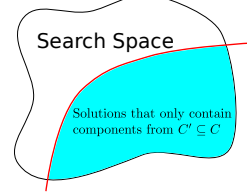
Differences between LNS and CMSA: summarized

How is the original problem instance reduced?

LNS



CMSA



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

Example: Repetition-free LCS problem

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

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

Principle: Builds a solution sequentially from left to right

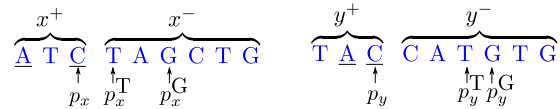
- 1: **input:** a problem instance (x, y, Σ)
- 2: **initialization:** $t := \epsilon$ (where ϵ is the empty string)
- 3: **while** $|\Sigma_t^{\text{nd}}| > 0$ **do**
- 4: $a := \text{ChooseFrom}(\Sigma_t^{\text{nd}})$
- 5: $t := ta$
- 6: **end while**
- 7: **output:** a repetition-free common subsequence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

Question: How is Σ_t^{nd} defined?

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

Example: Given is

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

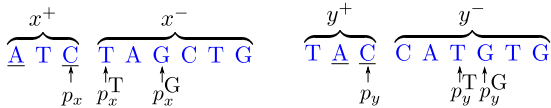


Result: $\Sigma_t^{\text{nd}} = \{T\}$

Greedy function

Greedy function:

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



ILP Model (1)

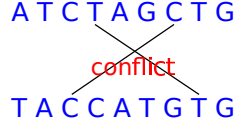
Set of binary variables:

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)

$$\begin{aligned} & \max \sum_{z_{i,j} \in Z} z_{i,j} \\ & \text{subject to:} \\ & \sum_{z_{i,j} \in Z_a} z_{i,j} \leq 1 \quad \text{for } a \in \Sigma \\ & z_{i,j} + z_{k,l} \leq 1 \quad \text{for all } z_{i,j} \text{ and } z_{k,l} \text{ being in conflict} \\ & z_{i,j} \in \{0, 1\} \quad \text{for } z_{i,j} \in Z \end{aligned} \quad \begin{matrix} (3) \\ (4) \\ (5) \\ (6) \end{matrix}$$

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:

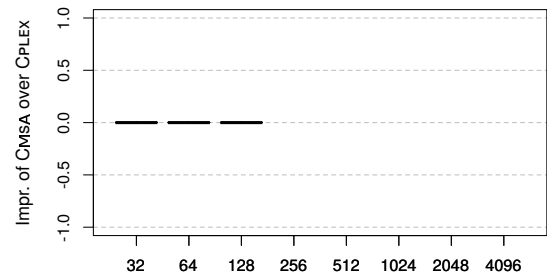
- ▶ Input sequence length: $n \in \{32, 64, 128, 256, 512, 1024, 2028, 4048\}$
- ▶ Alphabet size: $|\Sigma| \in \{n/8, n/4, 3n/8, n/2, 5n/8, 3n/4, 7n/8\}$

Set2:

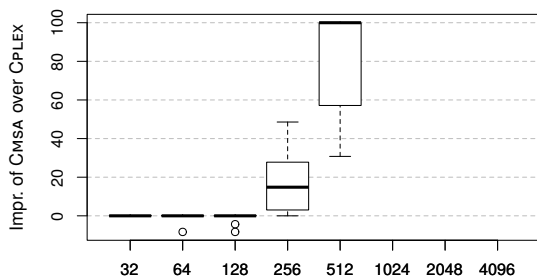
- ▶ Alphabet size: $|\Sigma| \in \{4, 8, 16, 32, 64, 128, 256, 512\}$
- ▶ Maximal number of repetitions of each letter: $rep \in \{3, 4, 5, 6, 7, 8\}$

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

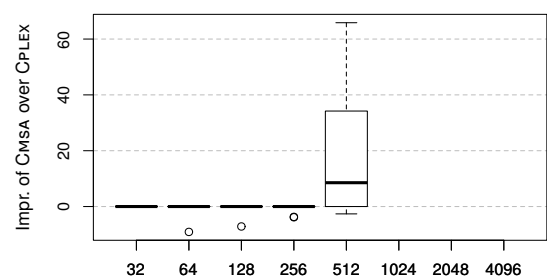
Experimental results: Set1

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

Experimental results: Set1

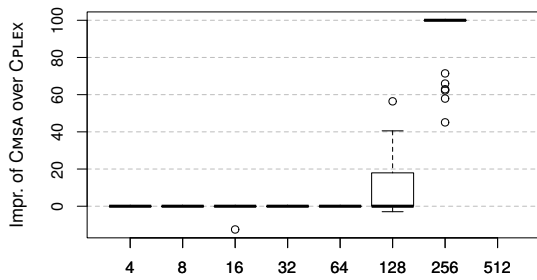
Improvement of CMSA over CPLEX: alphabet size $n/2$ 

Experimental results: Set1

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

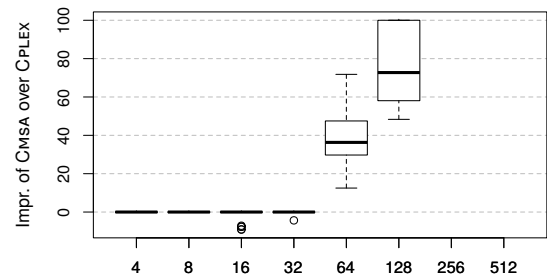
Experimental results: Set2

Improvement of CMSA over CPLEX: 3 reps



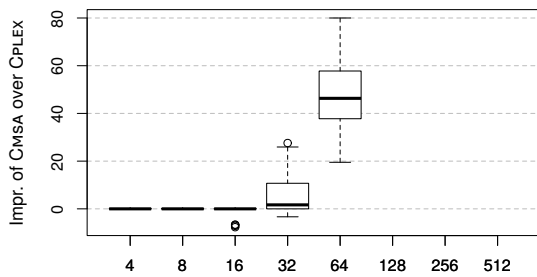
Experimental results: Set2

Improvement of CMSA over CPLEX: 6 reps

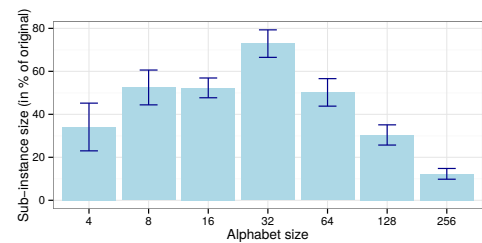


Experimental results: Set2

Improvement of CMSA over CPLEX: 8 reps



Experimental results: size of sub-instances



2nd Example: Minimum Common String Partition (1)

Given:

- Two **related sequences** of length n over alphabet Σ
- Note:** Sequences s_1 and s_2 are called **related** iff 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** iff $P_1 = P_2$
- Objective function:** $f(S) := |P_1| = |P_2|$

Objective: Minimization

Minimum Common String Partition (2)

Example:

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

Literature

Context:

- ▶ Introducido en 2005 en el contexto de la reordenación de genomas
- ▶ Complejidad: NP-duro

Related Works:

- ▶ 2005: Greedy algorithm
- ▶ 2007: approximation algorithm
- ▶ 2008: study on *fixed-parameter tractability (FPT)*
- ▶ 2013: ant colony optimization metaheuristic

Preliminaries

Definitions: given s_1 and s_2 ...

- ▶ A **common block** b_i is a triplet $(t_i, k1_i, k2_i)$ where
 1. t_i is a sequence starting at position $1 \leq k1_i \leq n$ in s_1
 2. t_i is a sequence 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: Set B of Common Blocks

Input sequences: $s_1 = \text{AGACTG}$ and $s_2 = \text{ACTAGG}$

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 $\{\text{ACT}, \text{AG}, \text{G}\}$: $S = \{b_1, b_2, b_{14}\}$

Observe: Set B is the set of all solution components

ILP Model (1)

Input sequences: $s_1 = \text{AGACTG}$ and $s_2 = \text{ACTAGG}$

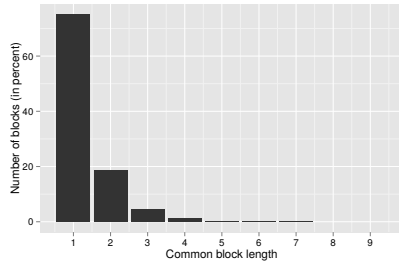
$$B = \left\{ \begin{array}{l} b_1 = (\text{ACT}, 3, 1) \\ b_2 = (\text{AG}, 1, 4) \\ b_3 = (\text{AC}, 3, 1) \\ b_4 = (\text{CT}, 4, 2) \\ b_5 = (\text{A}, 1, 1) \\ b_6 = (\text{A}, 1, 4) \\ b_7 = (\text{A}, 3, 1) \\ b_8 = (\text{A}, 3, 4) \\ b_9 = (\text{C}, 4, 2) \\ b_{10} = (\text{T}, 5, 3) \\ b_{11}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

ILP Model (2)

$$\begin{aligned} & \min \sum_{i=1}^m x_i & (7) \\ \text{subject to:} & \\ & \sum_{i=1}^m |t_i| \cdot x_i = n & (8) \\ & \sum_{i=1}^m M1_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \dots, n & (9) \\ & \sum_{i=1}^m M2_{i,j} \cdot x_i = 1 \quad \text{para } j = 1, \dots, n & (10) \\ & x_i \in \{0, 1\} \quad \text{para } i = 1, \dots, m \end{aligned}$$

Observe: This model has a very large number of variables (exponential)

Characteristics of Set B



Observe: the vast majority of common blocks of length 1 and 2 will not form part of good solutions

Simple Greedy Algorithm

Given a partial solution S_{partial} : $B(S_{\text{partial}}) \subset B$ can be used to extend S_{partial}

Pseudo-code:

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

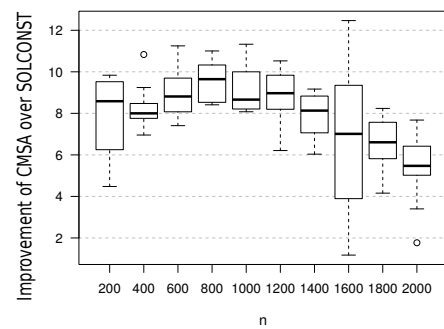
Note: This heuristic is used within CMSA in a **probabilistic** way

Problem Instances

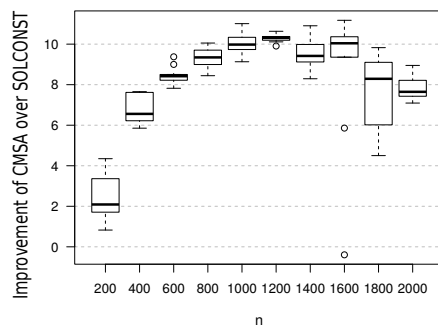
Instances: 300

- ▶ Length of the input sequences: $n \in \{200, 400, \dots, 1800, 2000\}$
- ▶ Alphabet sizes: $|\Sigma| \in \{4, 12, 20\}$

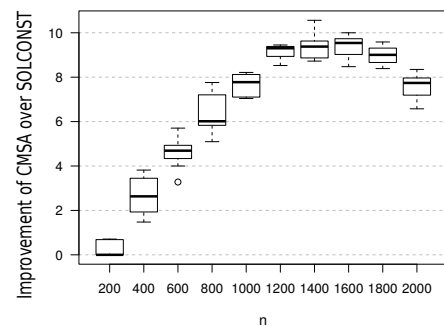
Results: Improvement over GREEDY ($|\Sigma| = 4$)

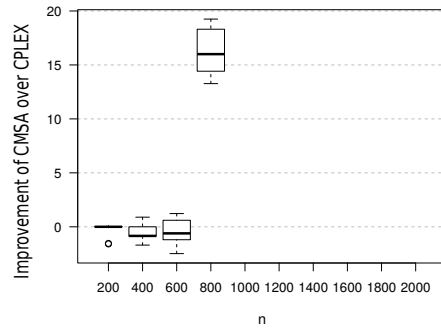
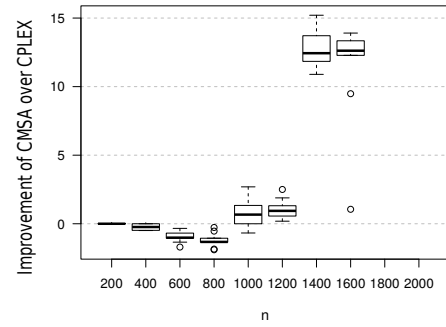


Results: Improvement over GREEDY ($|\Sigma| = 12$)

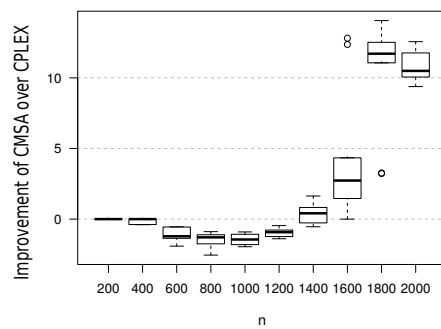


Results: Improvement over GREEDY ($|\Sigma| = 20$)



Results: Improvement over CPLEX ($|\Sigma| = 4$)Results: Improvement over CPLEX ($|\Sigma| = 12$)

x

Results: Improvement over CPLEX ($|\Sigma| = 20$)

Examples From My Recent Work

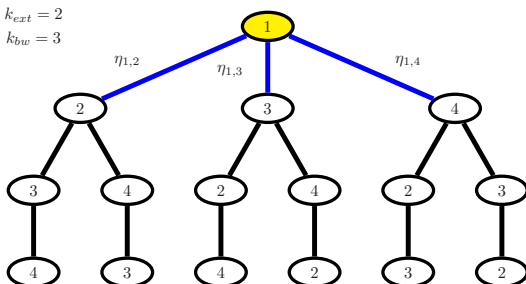
Beam-ACO: Combination between beam search and ant colony optimization

Beam search

1st construction step:

$$k_{ext} = 2$$

$$k_{bw} = 3$$

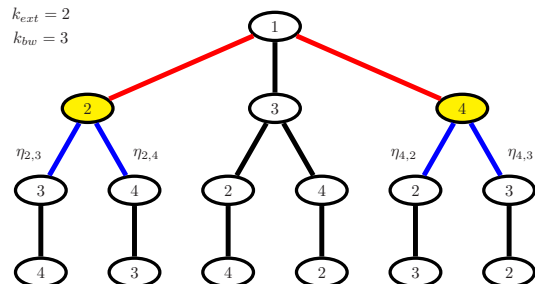


Beam search

2nd construction step:

$$k_{ext} = 2$$

$$k_{bw} = 3$$

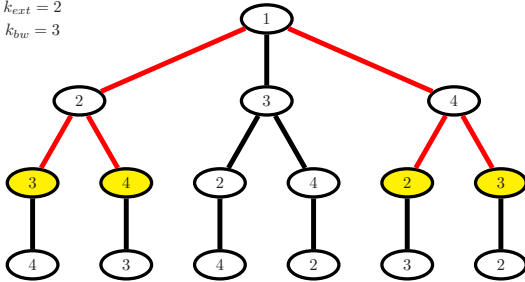


Beam search

After 2nd construction step: use **lower bound**

$$k_{ext} = 2$$

$$k_{bw} = 3$$

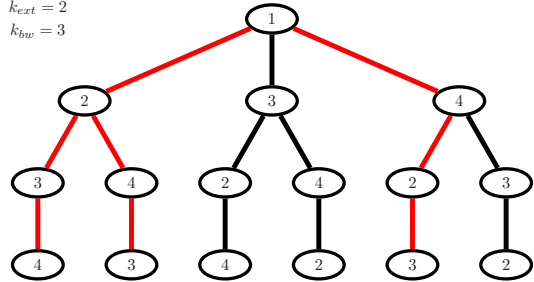


Beam search

3rd construction step:

$$k_{ext} = 2$$

$$k_{bw} = 3$$



Hybrid algorithm: Beam-ACO

Idea:

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

Advantages:

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

Hybrid algorithm: Beam-ACO

Applications **Beam-ACO** was applied to the following problems:

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

Hybrid algorithm: Beam-ACO

Question: Why does it work so well?

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

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

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

Summary and Conclusions

Presented topics:

- ▶ Hybrid metaheuristics: a short intro
- ▶ Despite criticism: term *hybrid metaheuristics* is useful
- ▶ Representative hybridization examples

Bottom line: More and more state-of-the-art methods are hybrids

But: There is still room for new, conceptually different hybrids!!

Questions?

Literature:

- ▶ G. R. Raidl. **Decomposition based hybrid metaheuristics**. *European Journal of Operational Research*, 2015
- ▶ 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



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