Construct, Merge, Solve & Adapt
A Hybrid Approach With a Resemblance to Evolutionary Algorithms

Christian Blum

ARTIFICIAL INTELLIGENCE RESEARCH INSTITUTE (IIIA)
SPANISH NATIONAL RESEARCH COUNCIL (CSIC)
Preliminaries: Preparing the Grounds

Optimization

Problems
- Combinatorial optimization problems
- Continuous optimization problems

Algorithms
- Metaheuristics
  - MHs based on solution construction
- Exact Techniques
  - ILP Solvers

Hybridization
Outline

- Introduction to Construct, Merge, Solve & Adapt (CMSA)

- Application example: Repetition-free longest common subsequence problem (RFLCS)

- Incorporating learning into CMSA

- A general CMSA for binary optimization
Why combining metaheuristics and ILP Solvers?

General advantage of metaheuristics:

▶ Very good in exploiting information on the problem (greedy heuristics)
▶ Very fast in deriving high-quality solutions

General advantage of ILP solvers:

▶ Based on cutting edge mathematical programming technologies
▶ Incredibly efficient up to a problem-specific instance size

Goal: Combine ILP solvers and (meta-)heuristics in order to use the power of ILP solvers in applications to larger problem instances
Standard: Large Neighborhood Search

► Small neighborhoods:

1. **Advantage:** It is fast to find an improving neighbor (if any)
2. **Disadvantage:** The average quality of the local minima is low

► Large neighborhoods:

1. **Advantage:** The average quality of the local minima is high
2. **Disadvantage:** Finding an improving neighbor might itself be \(NP\)-hard due to the size of the neighborhood

Ways of examining large neighborhoods:

► Heuristically

► **Exact techniques:** for example an ILP solver
ILP-based large neighborhood search: \textit{ILP-LNS}

1. Generate initial solution $S$
2. $S_{\text{partial}} := \text{Destroy } S \text{ partially}$
3. $S_{\text{ILP}} := \text{Apply ILP solver to } S_{\text{partial}}$
4. $S := \text{Choose between } S \text{ and } S_{\text{ILP}}$
Hypothesis and resulting research question

In our experience: LNS works especially well when

1. The **number of solution components** (variable-value combinations) is **not too high**

2. The **number of non-zero 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

- \( C \): complete set of solution components
- \( C' \subseteq C \): sub-instance
- Set the age of all \( c \in C \) to zero

- Probabilistically generate \( n_a \) solutions
  \( \hat{C} \): used solution components
- Add all \( c \in \hat{C} \setminus C' \) to \( C' \)
- \( S_{ILP} := \) Apply ILP solver to \( C' \)

- Increment age of all \( c \in C' \)
- Set age of all \( c \in S_{ILP} \) to zero
- Remove all \( c \in C' \setminus S_{ILP} \) that have reached the maximum age \( age_{max} \)
Resemblance to Evolutionary Algorithms?

- The approach evolves a small sub-instance which is supposed to contain high quality solutions.

- The application of the ILP solver at each iteration can be seen as optimal recombination.


- The ILP solver is responsible for the selection of the best solution components.
Differences between \textbf{LNS} and \textbf{CMSA}: summarized

How is the original problem instance reduced?

\textbf{LNS}:
- Partial destruction of the incumbent solution $S_p$

\textbf{CMSA}:
- Generating new solutions and removing \textbf{old} solution components

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

\begin{itemize}
  \item \textbf{LNS}: Partial destruction of the incumbent solution
  \item \textbf{CMSA}: Generating new solutions and removing \textbf{old} solution components
\end{itemize}
Example

Repetition-Free Longest Common Subsequence Problem
Longest common subsequence (LCS) problem (1)

Notation: What is a subsequence of a string?

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

Example: Is AAT a subsequence of ACAGTTA?

ACAGTTA
Longest common subsequence (LCS) problem (2)

**Problem definition (restricted to two input sequence)**

**Given:** A problem instance \((x, y, \Sigma)\), where

- \(x\) and \(y\) are input sequences over the alphabet \(\Sigma\)

**Optimization goal:**

Find a longest string \(t^*\) that is a subsequence of strings \(x\) and \(y\) → a longest common subsequence
**Repetition-free longest common subsequence problem**

- **Restriction:** No letter *may appear more than once* in a valid solution
- **Proposed in:** 2010 in *Discrete Applied Mathematics*
- **Hardness:** APX-hard (shown in above paper)
- **Motivation:** Genome rearrangement where duplicate genes are basically not considered
- **Existing algorithms:**
  1. Three simple heuristics, *Discrete Applied Mathematics*, 2010
A simple constructive RFLCS heuristic: Best-Next (1)

**Principle:** Builds a solution sequentially from left to right

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

**Question:** How is \(\Sigma_t^{nd}\) defined?
A simple constructive LCS heuristic: Best-Next (2)

Example: Given is

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

- **Partial solution** $t = AC$

Result: $\Sigma^nd_t = \{T\}$
Greedy function

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

\[
\begin{array}{cccccccc}
A & T & C & T & A & G & C & T & G \\
T & A & C & C & A & T & G & T & G \\
\end{array}
\]

Example of a conflict

\[
\begin{array}{cccccccc}
A & T & C & T & A & G & C & T & G \\
T & A & C & C & A & T & G & T & G \\
\end{array}
\]

conflict
ILP Model (2)

\[
\begin{align*}
\max & \quad \sum_{z_{i,j} \in Z} z_{i,j} \\
\text{subject to:} & \\
\sum_{z_{i,j} \in Z_a} z_{i,j} & \leq 1 \quad \text{for } a \in \Sigma \\
\sum_{z_{i,j} \in Z_a} z_{i,j} + z_{k,l} & \leq 1 \quad \text{for all } z_{i,j} \text{ and } z_{k,l} \text{ being in conflict} \\
z_{i,j} & \in \{0,1\} \quad \text{for } z_{i,j} \in Z
\end{align*}
\]

Hereby:

\[
\begin{align*}
\text{\textbullet} & \quad z_{i,j} \in Z_a \text{ iff } x[i] = y[j] = a \\
\text{\textbullet} & \quad z_{i,j} \text{ and } z_{k,l} \text{ are in conflict iff } i < k \text{ and } j > l \text{ OR } i > k \text{ and } j < l
\end{align*}
\]
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
Using Learning for The Generation of Solutions

First results
General Idea

Current applications of CMSA:

- Probabilistically construct $n$ solutions at each iteration
- The probability distribution is always the same

Idea: Can we use some sort of learning in order to sample solutions from promising parts of the search space?

First approach: Make use of ant colony optimization
Ant Colony Optimization (ACO): Schematic View

Generally biased by pheromone values and greedy information
CMSA-Heur with Solution Constructions in Parallel

- Probab. Heuristic
- Application of CPLEX to Subinst.
- CPLEX uses n processors
- Probab. Heuristic
- Application of CPLEX to Subinst.
- CPLEX uses n processors

n solution constructions in parallel
CMSA-ACO Running Colonies in Parallel

Run n applications of ACO in parallel

CPLEX uses n processors

Start

Application of CPLEX to Subinst.

Applic. of ACO

Application of CPLEX to Subinst.

Applic. of ACO

Applic. of ACO

Finish
Experimental results: Instance group 4 (largest)
A Problem-Agnostic CMSA for Binary Programming

First results
Initial Considerations

\[ \min \{ \vec{c}^T \vec{x} : A\vec{x} \leq \vec{b}, x_j \in \{0, 1\} \ \forall j = 1, \ldots, n \} \quad (5) \]

where

- \( A \) is an \( m \times n \) matrix
- \( \vec{b} \) is the right-hand-size vector of size \( m \)
- \( \vec{c} \) is a cost vector
- \( \vec{x} \) is the vector of \( n \) binary decision variables

Before algorithm start:

1. Run node heuristic of CPLEX to find a first solution \( \vec{s}^{bsf} \)
2. If none found: solve the LP relaxation \( \vec{x}^{LP} \)
Main Challenge: Solution Construction

Step 1: Generate sample vector $\vec{x}_{\text{samp}}$

If $\vec{s}_{\text{bsf}} \neq \text{NULL}$: make use of $\vec{s}_{\text{bsf}}$ and determinism rate $0 < d_{\text{rate}} < 0.5$

$$x_{\text{samp}}^j = \begin{cases} d_{\text{rate}} & \text{if } s_{j}^{\text{bsf}} = 0 \\ 1 - d_{\text{rate}} & \text{if } s_{j}^{\text{bsf}} = 1 \end{cases}$$

for all $j = 1, \ldots, n$.

Otherwise: make use of $\vec{x}_{\text{LP}}$

$$x_{\text{samp}}^j = \begin{cases} x_{j}^{\text{LP}} & \text{if } d_{\text{rate}} \leq x_{j}^{\text{LP}} \leq 1 - d_{\text{rate}} \\ d_{\text{rate}} & \text{if } x_{j}^{\text{LP}} < d_{\text{rate}} \\ 1 - d_{\text{rate}} & \text{if } x_{j}^{\text{LP}} > 1 - d_{\text{rate}} \end{cases}$$
Results: Instance air04 (class: easy)
Results: Instance rmine14 (class: open)
Results: Instance 2048_3n-div-8_0 (class: large-scale)
Summary and Possible Research Directions

Summary:

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

Possible Research Directions:

▶ Solution construction: adaptive probabilities over time
▶ A more intelligent version of the aging mechanism
▶ Developing a problem-agnostic CMSA for general ILPs
Questions?

Literature:

