# A Hybrid Heuristic for the Multi-objective Knapsack Problem

## Doctoral thesis proposal

Marcos Daniel V. Baroni

Universidade Federal do Espírito Santo
Departamento de Informática
Programa de Pós-graduação em Informática

27 de novembro de 2017

# Table of Contents

## Multi-objective problems

**Multi-objective problems** are optimization problems involving optimizing multiple criteria.

## Multi-objective problems

**Multi-objective problems** are optimization problems involving optimizing multiple criteria.

- Usually conflicting criterias;

# Multi-objective problems

**Multi-objective problems** are optimization problems involving optimizing multiple criteria.

- Usually conflicting criterias;
- A set of *optimal* (non-dominated) solutions;

## Multi-objective problems

**Multi-objective problems** are optimization problems involving optimizing multiple criteria.

- Usually conflicting criterias;
- A set of *optimal* (non-dominated) solutions;
- Models many real applications.

## The multi-dimensional knapsack problem

The **multi-dimensional knapsack problem** (MOKP) is one of the most important multi-objective problems.

## The multi-dimensional knapsack problem

The **multi-dimensional knapsack problem** (MOKP) is one of the most important multi-objective problems.

- Is a generalization of the $0-1$ knapsack problem ($\mathcal{NP}$-Hard);

# The multi-dimensional knapsack problem

The **multi-dimensional knapsack problem** (MOKP) is one of the most important multi-objective problems.

- Is a generalization of the $0-1$ knapsack problem ($\mathcal{NP}$-Hard);
- Models project selection, capital budgeting, cargo loading, flow shop scheduling and others;

## The multi-dimensional knapsack problem

The **multi-dimensional knapsack problem** (MOKP) is one of the most important multi-objective problems.

- Is a generalization of the $0-1$ knapsack problem ($\mathcal{NP}$-Hard);
- Models project selection, capital budgeting, cargo loading, flow shop scheduling and others;
- Models many real applications;

## Motivation

Several exact approaches have been proposed in the literature for solving the MOKP.

## Motivation

Several exact approaches have been proposed in the literature for solving the MOKP.

However no effective exact method for large MOKP, expecially with more than two objectives.

## Motivation

Several exact approaches have been proposed in the literature for solving the MOKP.

However no effective exact method for large MOKP, expecially with more than two objectives.

Even for the bi-objective case, some medium sized instances has shown to be hard to solve exactly.

# Motivation

Several exact approaches have been proposed in the literature for solving the MOKP.

However no effective exact method for large MOKP, expecially with more than two objectives.

Even for the bi-objective case, some medium sized instances has shown to be hard to solve exactly.

This reason motivates the development of heuristic methods.

## Proposal

This work proposes the development of a heuristic for the MOKP based on an evolutionary algorithm called shuffled complex evolution (SCE).

## Proposal

This work proposes the development of a heuristic for the MOKP based on an evolutionary algorithm called shuffled complex evolution (SCE).

As a performance improvement for the approach, a multi-dimensional indexing strategy will be used for handling the large amount of solutions.

## Proposal

This work proposes the development of a heuristic for the MOKP based on an evolutionary algorithm called shuffled complex evolution (SCE).

As a performance improvement for the approach, a multi-dimensional indexing strategy will be used for handling the large amount of solutions.

The SCE has been successfully used for solving optimization problems, among then, the multi-dimensional knapsack problem (MKP) (also a contribution of this work).

# Contributions and Publications

## Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**

## Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
    - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.

# Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
    - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.

2. **A SCE algorithm for the MKP:**

# Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
   - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.
2. **A SCE algorithm for the MKP:**
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications.* [S.l.]: Springer, 2015. p. 768-775.

# Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
   - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.
2. **A SCE algorithm for the MKP:**
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications.* [S.l.]: Springer, 2015. p. 768-775.
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In: IEEE. *Evolutionary Computation (CEC), 2016 IEEE Congress on.* [S.l.], 2016 p. 2718-2723.

# Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
   - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.
2. **A SCE algorithm for the MKP:**
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications.* [S.l.]: Springer, 2015. p. 768-775.
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In: IEEE. *Evolutionary Computation (CEC), 2016 IEEE Congress on.* [S.l.], 2016 p. 2718-2723.
3. **Hybrid heuristic for MOKP:**

# Contributions and Publications

1. **Efficient indexing strategy for MOKP solutions:**
   - BARONI, M. D. V; VAREJÃO, F. M. Multi-dimensional indexing on dynamic programming for multi-objective knapsack problem. *International Transactions in Operational Research*, Wiley Online Library, 2017, Submitted.
2. **A SCE algorithm for the MKP:**
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications.* [S.l.]: Springer, 2015. p. 768-775.
   - BARONI, M. D. V; VAREJÃO, F. M. A shuffled complex evolution algorithm for the multidimensional knapsack problem using core concept. In: IEEE. *Evolutionary Computation (CEC), 2016 IEEE Congress on.* [S.l.], 2016 p. 2718-2723.
3. **Hybrid heuristic for MOKP:**
   - BARONI, M. D. V; VAREJÃO, F. M. An efficient shuffled complex evolution algorithm for the multi-objective knapsack problem. In: IEEE *Evolutionary Computation (CEC), 2018 IEEE Congress on.* [S.l.], 2018, In Preparation

# The multi-objective knapsack problem

## Multi-objective optimization problems

### Definition

A general multi-objective optimization problem can be described as a vector function $f$ that maps a decision variable (solution) to a tuple of $m$ objectives.

## Multi-objective optimization problems

### Definition

A general multi-objective optimization problem can be described as a vector function $f$ that maps a decision variable (solution) to a tuple of $m$ objectives.

$$\max \boldsymbol{y} = f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big)$$

$$\text{subject to } \boldsymbol{x} \in X$$

## Multi-objective optimization problems

### Definition

A general multi-objective optimization problem can be described as a vector function $f$ that maps a decision variable (solution) to a tuple of $m$ objectives.

$$\max\ \boldsymbol{y} = f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big)$$
$$\text{subject to } \boldsymbol{x} \in X$$

$\boldsymbol{x}$ is the decision variable;

# Multi-objective optimization problems

## Definition

A general multi-objective optimization problem can be described as a vector function $f$ that maps a decision variable (solution) to a tuple of $m$ objectives.

$$\max \boldsymbol{y} = f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big)$$
$$\text{subject to } \boldsymbol{x} \in X$$

$\boldsymbol{x}$ is the decision variable;
$X$ denotes the set of feasible solutions;

# Multi-objective optimization problems

## Definition

A general multi-objective optimization problem can be described as a vector function $f$ that maps a decision variable (solution) to a tuple of $m$ objectives.

$$\max \ \boldsymbol{y} = f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big)$$
$$\text{subject to } \boldsymbol{x} \in X$$

$\boldsymbol{x}$ is the decision variable;
$X$ denotes the set of feasible solutions;
$\boldsymbol{y}$ is the objective vector, each one has to be maximized.

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$.

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives

## Multi-objective optimization problems

### Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

$$dom(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(\boldsymbol{a}) \geqslant f_i(\boldsymbol{b}) \text{ and} \\ \exists j \in \{1, 2, \ldots, m\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

$$dom(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(\boldsymbol{a}) \geqslant f_i(\boldsymbol{b}) \text{ and} \\ \exists j \in \{1, 2, \ldots, m\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

## Efficiency

A feasible solution $\boldsymbol{x} \in X$ is called efficient if it is not dominated by any other feasible solution.

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

$$dom(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(\boldsymbol{a}) \geqslant f_i(\boldsymbol{b}) \text{ and} \\ \exists j \in \{1, 2, \ldots, m\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

## Efficiency

A feasible solution $\boldsymbol{x} \in X$ is called efficient if it is not dominated by any other feasible solution.

$$efficient(\boldsymbol{x}) = \nexists \boldsymbol{a} \in X : dom(\boldsymbol{a}, \boldsymbol{x})$$

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

$$dom(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(\boldsymbol{a}) \geqslant f_i(\boldsymbol{b}) \text{ and} \\ \exists j \in \{1, 2, \ldots, m\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

## Efficiency

A feasible solution $\boldsymbol{x} \in X$ is called efficient if it is not dominated by any other feasible solution.

$$efficient(\boldsymbol{x}) = \nexists \boldsymbol{a} \in X : dom(\boldsymbol{a}, \boldsymbol{x})$$

## Pareto set

The set of all efficient solutions.

# Multi-objective optimization problems

## Dominance

Consider two decision vectors $\boldsymbol{a}, \boldsymbol{b} \in X$. Vector $\boldsymbol{a}$ dominates $\boldsymbol{b}$ if, and only if $\boldsymbol{a}$ is at least as good as $\boldsymbol{b}$ in all objectives and better than $\boldsymbol{b}$ in at least one objective.

$$dom(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} \forall i \in \{1, 2, \ldots, m\} : f_i(\boldsymbol{a}) \geqslant f_i(\boldsymbol{b}) \text{ and} \\ \exists j \in \{1, 2, \ldots, m\} : f_j(\boldsymbol{a}) > f_j(\boldsymbol{b}) \end{cases}$$

## Efficiency

A feasible solution $\boldsymbol{x} \in X$ is called efficient if it is not dominated by any other feasible solution.

$$efficient(\boldsymbol{x}) = \nexists \boldsymbol{a} \in X : dom(\boldsymbol{a}, \boldsymbol{x})$$

## Pareto set

The set of all efficient solutions.

$$Pareto(X) = \{\boldsymbol{x} \in X : efficient(\boldsymbol{x})\}$$

# Multi-objective optimization problems

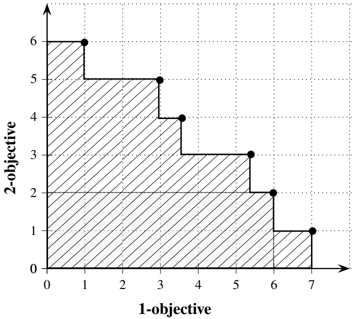



Figure: Example of a Pareto set for a bi-objective problem.

# The multi-objective knapsack problem

## Problem Definition

A problem instance with $n$ items and $m$ objectives:

$$\begin{aligned} \max\ f(\boldsymbol{x}) &= \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big) \\ \text{subject to } w(\boldsymbol{x}) &\leqslant W \\ \boldsymbol{x} &\subseteq \{1, \ldots, n\} \end{aligned}$$

# The multi-objective knapsack problem

## Problem Definition

A problem instance with $n$ items and $m$ objectives:

$$\begin{aligned} \max\ f(\boldsymbol{x}) &= \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big) \\ \text{subject to } w(\boldsymbol{x}) &\leqslant W \\ \boldsymbol{x} &\subseteq \{1, \ldots, n\} \end{aligned}$$

where

$$f_j(\boldsymbol{x}) = \sum_{i \in \boldsymbol{x}} p_i^j \quad j = 1, \ldots, m$$

$$w(\boldsymbol{x}) = \sum_{i \in \boldsymbol{x}} w_i$$

# The multi-objective knapsack problem

## Problem Definition

A problem instance with $n$ items and $m$ objectives:

$$\max \; f(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})\big)$$

$$\text{subject to } w(\boldsymbol{x}) \leqslant W$$

$$\boldsymbol{x} \subseteq \{1, \ldots, n\}$$

where

$$f_j(\boldsymbol{x}) = \sum_{i \in \boldsymbol{x}} p_i^j \quad j = 1, \ldots, m$$

$$w(\boldsymbol{x}) = \sum_{i \in \boldsymbol{x}} w_i$$

A $\mathcal{NP}$-Hard problem for which the Pareto optimal set tends to grow exponentially with the number of objectives.

# The multi-objective knapsack problem

## Knapsack dominance

Consider two solutions $\boldsymbol{x}, \boldsymbol{y}$ for a MOKP.

We say $\boldsymbol{x}$ knapsack-dominates $\boldsymbol{y}$, denoted as $dom_k(\boldsymbol{x}, \boldsymbol{y})$, if $\boldsymbol{x}$ dominates $\boldsymbol{y}$ and does not weight more than $\boldsymbol{y}$.

# The multi-objective knapsack problem

## Knapsack dominance

Consider two solutions $\boldsymbol{x}, \boldsymbol{y}$ for a MOKP.

We say $\boldsymbol{x}$ knapsack-dominates $\boldsymbol{y}$, denoted as $dom_k(\boldsymbol{x}, \boldsymbol{y})$, if $\boldsymbol{x}$ dominates $\boldsymbol{y}$ and does not weight more than $\boldsymbol{y}$. Formally:

$$dom_k(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} dom(\boldsymbol{x}, \boldsymbol{y}) & \text{and} \\ w(\boldsymbol{x}) \leqslant w(\boldsymbol{y}) \end{cases}$$

# The multi-objective knapsack problem

## Knapsack dominance

Consider two solutions $\boldsymbol{x}, \boldsymbol{y}$ for a MOKP.

We say $\boldsymbol{x}$ knapsack-dominates $\boldsymbol{y}$, denoted as $dom_k(\boldsymbol{x}, \boldsymbol{y})$, if $\boldsymbol{x}$ dominates $\boldsymbol{y}$ and does not weight more than $\boldsymbol{y}$. Formally:

$$dom_k(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} dom(\boldsymbol{x}, \boldsymbol{y}) & \text{and} \\ w(\boldsymbol{x}) \leqslant w(\boldsymbol{y}) \end{cases}$$

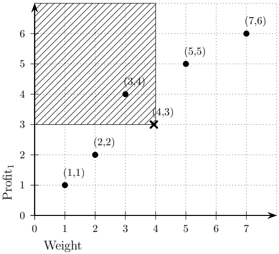



Figure: A knapsack-dominated solution.

# Multi-dimensional Indexing of MOKP solutions

## Dominance check operation

During the process of solving a MOKP or other multi-objective problems one of the main general operation is to check if a solution is dominated by another.

## Dominance check operation

During the process of solving a MOKP or other multi-objective problems one of the main general operation is to check if a solution is dominated by another.

An algorithm usually requires selecting all non-dominated solutions from a large set of partial solutions.

## Dominance check operation

During the process of solving a MOKP or other multi-objective problems one of the main general operation is to check if a solution is dominated by another.

An algorithm usually requires selecting all non-dominated solutions from a large set of partial solutions.

This may demand quadratic effort on total number of solution if implemented as pairwise comparison.

# Dominance check operation

During the process of solving a MOKP or other multi-objective problems one of the main general operation is to check if a solution is dominated by another.

An algorithm usually requires selecting all non-dominated solutions from a large set of partial solutions.

This may demand quadratic effort on total number of solution if implemented as pairwise comparison.

However, if solutions are mapped into points in a multi-dimensional space, this operation corresponds on checking whether a point exists in a certain region.

# Dominance check operation

Formally:

$$\text{if } \ dom_k(\boldsymbol{y}, \boldsymbol{x}) \ \text{ then } \ pnt(\boldsymbol{y}) \in R(\boldsymbol{x})$$

where

$$pnt(\boldsymbol{x}) = \Big(f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}), w(\boldsymbol{x})\Big)$$

$$R(\boldsymbol{x}) = \Big\{ a \in \mathbb{R}^{m+1} \ \Big| \ a_{m+1} \leqslant w(\boldsymbol{x}) \ \text{ and } \ a_i \geqslant f_i(\boldsymbol{x}), \ i \in \{1, \ldots, m\} \Big\}$$

## Dominance check operation

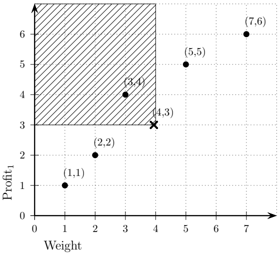
Formally:

$$\text{if } dom_k(\boldsymbol{y}, \boldsymbol{x}) \text{ then } pnt(\boldsymbol{y}) \in R(\boldsymbol{x})$$

where

$$pnt(\boldsymbol{x}) = \Big(f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}), w(\boldsymbol{x})\Big)$$

$$R(\boldsymbol{x}) = \Big\{ a \in \mathbb{R}^{m+1} \;\Big|\; a_{m+1} \leqslant w(\boldsymbol{x}) \text{ and } a_i \geqslant f_i(\boldsymbol{x}),\ i \in \{1, \ldots, m\} \Big\}$$



## Multidimensional solution indexing

The problem of determining whether a point exists in a certain region of space is known as range search.

## Multidimensional solution indexing

The problem of determining whether a point exists in a certain region of space is known as range search.

For efficiency reason range search operations is usually executed with the assistance of a *k*-d tree.

# Multidimensional solution indexing

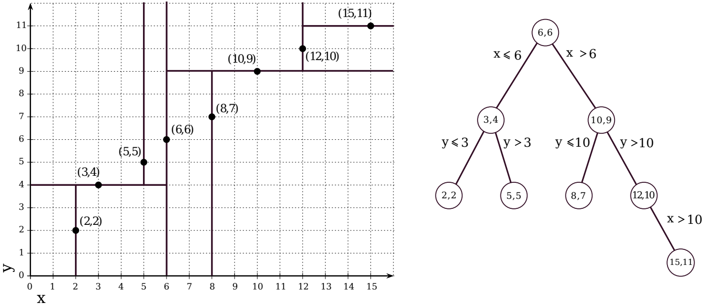
The problem of determining whether a point exists in a certain region of space is known as range search.

For efficiency reason range search operations is usually executed with the assistance of a *k*-d tree.

The *k*-d tree is a type of binary search tree for indexing multi-dimensional data with simple construction and low space usage.

# Multidimensional solution indexing

Unlike a standard binary tree, that uses only one key for all levels of the tree, the *k*-d tree uses *k* keys and cycles through these keys for successive levels of the tree.



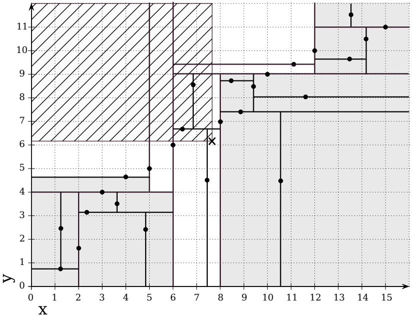

(a) points displayed on a plane

(b) points arranged on a kd-tree

Figure: Example of points indexed in a *k*-d tree.

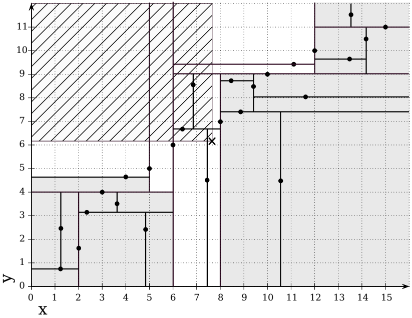
# Multidimensional solution indexing

Example of dominance check operation using $k$-d tree:



# Multidimensional solution indexing

Example of dominance check operation using $k$-d tree:




The efficiency of this pruning action grows with the amount of points.

## Multi-dimensional Indexing of MOKP solutions

Use case on an exact method

## Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

# Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

## Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

1: **function** $\mathrm{DP}(\boldsymbol{p}, \boldsymbol{w}, W)$

## Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

**function** $\mathrm{DP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \left\{\emptyset\right\}$

# Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

**function** $\mathrm{DP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \left\{\emptyset\right\}$
    **for** $k \leftarrow 1, n$ **do**
        $S_*^k = S^{k-1} \cup \{\boldsymbol{x} \cup k \mid \boldsymbol{x} \in S^{k-1}\}$    ▷ solutions extension

# Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

```
function DP(p, w, W)
    S^0 = {∅}
    for k ← 1, n do
        S^k_* = S^{k−1} ∪ {x ∪ k | x ∈ S^{k−1}}          ▷ solutions extension
        S^k = {x | ∄a ∈ S^k_* : dom_k(a, x)}            ▷ partial dominance filter
    end for
```

## Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

**function** $\mathrm{DP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \left\{\emptyset\right\}$
    **for** $k \leftarrow 1, n$ **do**
        $S^k_* = S^{k-1} \cup \{\boldsymbol{x} \cup k \mid \boldsymbol{x} \in S^{k-1}\}$ ▷ solutions extension
        $S^k = \{\boldsymbol{x} \mid \nexists \boldsymbol{a} \in S^k_* : dom_k(\boldsymbol{a}, \boldsymbol{x})\}$ ▷ partial dominance filter
    **end for**
    $P = \{\boldsymbol{x} \mid \nexists \boldsymbol{a} \in S^n : dom(\boldsymbol{a}, \boldsymbol{x}) \mid w(\boldsymbol{x}) \leqslant W\}$ ▷ dominance/feasibility

## Nemhauser and Ullmann's algorithm

As an application of the indexing proposal we will use the $k$-d tree in an *state-of-art* exact dynamic programming algorithm for the MOKP.

The algorithm can be seen as a MOKP specialization of the classical Nemhauser and Ullmann's algorithm for generically solving knapsack problems.

**function** $\mathrm{DP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \{\emptyset\}$
    **for** $k \leftarrow 1, n$ **do**
        $S^k_* = S^{k-1} \cup \{\boldsymbol{x} \cup k \mid \boldsymbol{x} \in S^{k-1}\}$ ▷ solutions extension
        $S^k = \{\boldsymbol{x} \mid \nexists \boldsymbol{a} \in S^k_* : dom_k(\boldsymbol{a}, \boldsymbol{x})\}$ ▷ partial dominance filter
    **end for**
    $P = \{\boldsymbol{x} \mid \nexists \boldsymbol{a} \in S^n : dom(\boldsymbol{a}, \boldsymbol{x}) \mid w(\boldsymbol{x}) \leqslant W\}$ ▷ dominance/feasibility
    **return** $P$
**end function**

# Bazgan's dynamic programming algorithm

**function** $\textsc{BazDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \left\{\emptyset\right\}$

# Bazgan's dynamic programming algorithm

Item reordering

1: **function** $\textsc{BazDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
2: $S^0 = \left\{\emptyset\right\}$
3: $o_1, \ldots, o_n = \mathcal{O}^{max}$

## Bazgan's dynamic programming algorithm

**function** $\textsc{BazDP}(\boldsymbol{p}, \boldsymbol{w}, W)$

$\quad S^0 = \left\{\emptyset\right\}$

$\quad o_1, \ldots, o_n = \mathcal{O}^{max}$

$\quad$ **for** $k \leftarrow 1, n$ **do**

# Bazgan's dynamic programming algorithm

Feasible extension

1: **function** BAZDP($\boldsymbol{p}, \boldsymbol{w}, W$)
2: $S^0 = \left\{\emptyset\right\}$
3: $o_1, \ldots, o_n = \mathcal{O}^{max}$
4: **for** $k \leftarrow 1, n$ **do**
5: $S_*^k = \left\{\boldsymbol{x} \cup \{o_k\} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} \leqslant W\right\}$

# Bazgan's dynamic programming algorithm

Defficiency avoidance

1: **function** $\text{BAZDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
2: $S^0 = \{\emptyset\}$
3: $o_1, \ldots, o_n = \mathcal{O}^{max}$
4: **for** $k \leftarrow 1, n$ **do**
5: $S_*^k = \left\{ \boldsymbol{x} \cup \{o_k\} \,\middle|\, \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} \leqslant W \right\}$
6: $\cup \left\{ \boldsymbol{x} \,\middle|\, \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} + \ldots + w_{o_n} > W \right\}$

# Bazgan's dynamic programming algorithm

Unpromising elimination

**function** $\text{BAZDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
$S^0 = \left\{\emptyset\right\}$
$o_1, \ldots, o_n = \mathcal{O}^{max}$
**for** $k \leftarrow 1, n$ **do**
$S^k_* = \left\{\boldsymbol{x} \cup \{o_k\} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} \leqslant W\right\}$
$\cup \left\{\boldsymbol{x} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} + \ldots + w_{o_n} > W\right\}$
$S^k = \left\{\boldsymbol{x} \in S^k_* \;\middle|\; (\nexists \boldsymbol{a} \in S^k_*)\left[dom_k(\boldsymbol{a}, \boldsymbol{x}) \vee dom\Big(lb(\boldsymbol{a}), ub(\boldsymbol{x})\Big)\right]\right\}$

# Bazgan's dynamic programming algorithm

**function** $\textsc{BazDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
    $S^0 = \left\{\emptyset\right\}$
    $o_1, \ldots, o_n = \mathcal{O}^{max}$
    **for** $k \leftarrow 1, n$ **do**
        $S_*^k = \left\{\boldsymbol{x} \cup \{o_k\} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} \leqslant W\right\}$
            $\cup \left\{\boldsymbol{x} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} + \ldots + w_{o_n} > W\right\}$
        $S^k = \left\{\boldsymbol{x} \in S_*^k \;\middle|\; (\nexists \boldsymbol{a} \in S_*^k)\left[dom_k(\boldsymbol{a}, \boldsymbol{x}) \vee dom\Big(lb(\boldsymbol{a}), ub(\boldsymbol{x})\Big)\right]\right\}$
    **end for**
    **return** $S^n$
**end function**

# Bazgan's dynamic programming algorithm

Dominance check operation

**function** $\textsc{BazDP}(\boldsymbol{p}, \boldsymbol{w}, W)$
$\quad S^0 = \left\{\emptyset\right\}$
$\quad o_1, \ldots, o_n = \mathcal{O}^{max}$
$\quad$ **for** $k \leftarrow 1, n$ **do**
$$S^k_* = \left\{\boldsymbol{x} \cup \{o_k\} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} \leqslant W\right\}$$
$$\cup \left\{\boldsymbol{x} \;\middle|\; \boldsymbol{x} \in S^{k-1} \wedge w(\boldsymbol{x}) + w_{o_k} + \ldots + w_{o_n} > W\right\}$$
$$S^k = \left\{\boldsymbol{x} \in S^k_* \;\middle|\; (\nexists \boldsymbol{a} \in S^k_*)\left[\underline{dom_k}(\boldsymbol{a}, \boldsymbol{x}) \vee \underline{dom}\big(lb(\boldsymbol{a}), ub(\boldsymbol{x})\big)\right]\right\}$$
$\quad$ **end for**
$\quad$ **return** $S^n$
**end function**

## Computational Experiments

Computational experiments on bi-dimensional instances:

A) Random instances:
$p_i^j \in [1, 1000]$,
$w_i \in [1, 1000]$.

B) Unconflicting instances:
$p_i^1 \in [111, 1000]$,
$p_i^2 \in [p_i^1 - 100, p_i^1 + 100]$,
$w_i \in [1, 1000]$.

C) Conflicting instances:
$p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [1, 1000]$.

D) Conflicting instances with correlated weight:
$p_i^1 \in [1, 1000]$,
$p_i^2 \in [max\{900 - p_i^1; 1\}, min\{1100 - p_i^1, 1000\}]$,
$w_i \in [p_i^1 + p_i^2 - 200, p_i^1 + p_i^2 + 200]$.

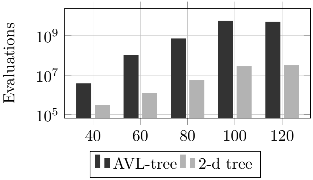
## Computational Experiments

Average CPU-time for bi-objective instances:

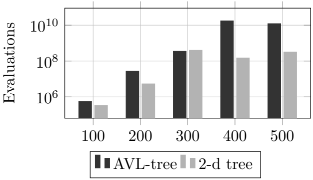
| Instance | | | AVL tree | 2-d tree | |
|---|---|---|---|---|---|
| Type | $n$ | $\|ND\|$ | time (s) | time (s) | speedup |
| A | 40 | 38.1 | **0.06** | **0.06** | 1.0 |
| | 60 | 73.1 | 1.12 | **0.88** | 1.3 |
| | 80 | 125.6 | 19.81 | **11.89** | 1.7 |
| | 100 | 180.4 | 165.24 | **76.50** | 2.2 |
| | 120 | 233.9 | 708.53 | **361.87** | 2.0 |
| B | 100 | 3.1 | **0.02** | 0.08 | 0.3 |
| | 200 | 10.0 | **0.80** | 5.09 | 0.2 |
| | 300 | 24.9 | **9.45** | 88.30 | 0.1 |
| | 400 | 36.2 | **95.39** | 730.04 | 0.1 |
| | 500 | 53.7 | **255.57** | 2824.65 | 0.1 |
| C | 20 | 36.6 | **0.00** | **0.00** | 1.0 |
| | 40 | 102.8 | 0.65 | **0.42** | 1.5 |
| | 60 | 231.9 | 28.98 | **14.09** | 2.1 |
| | 80 | 358.0 | 564.10 | **241.54** | 2.3 |
| | 100 | 513.8 | 3756.57 | **1605.19** | 2.3 |
| D | 20 | 174.9 | 0.15 | **0.12** | 1.3 |
| | 30 | 269.3 | 16.82 | **7.60** | 2.2 |
| | 40 | 478.0 | 395.76 | **186.67** | 2.1 |
| | 50 | 553.4 | 2459.48 | **1417.94** | 1.7 |

# Computational Experiments

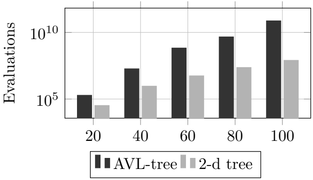
Average number of solution evaluations for bi-objective instances:



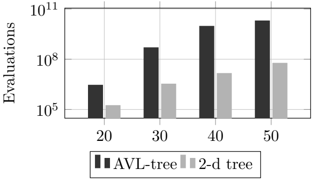

(a) Type A




(b) Type B




(c) Type C




(d) Type D

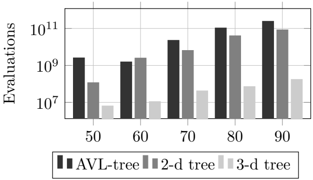
## Computational Experiments

Average CPU-time for 3-objective instances:

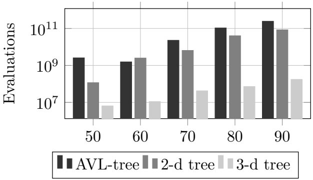
| Instance | | | AVL tree | 2-d tree | | 3-d tree | |
|---|---|---|---|---|---|---|---|
| Type | $n$ | $\|ND\|$ | time (s) | time (s) | speedup | time (s) | speedup |
| A | 50 | 557.5 | 41.2 | 21.3 | 1.9 | **18.5** | 2.2 |
| | 60 | 1240.0 | 485.9 | 247.8 | 1.9 | **79.9** | 6.0 |
| | 70 | 1879.3 | 3179.5 | 1038.0 | 3.0 | **614.5** | 5.1 |
| | 80 | 2540.5 | 6667.9 | 3796.0 | 1.7 | **2943.9** | 2.2 |
| | 90 | 3528.5 | 24476.5 | 12916.7 | 1.8 | **3683.7** | 6.6 |
| B | 100 | 18.0 | **0.1** | 0.3 | 0.3 | 0.3 | 0.3 |
| | 200 | 65.4 | **11.4** | 34.4 | 0.3 | 29.1 | 0.4 |
| | 300 | 214.2 | **307.7** | 631.5 | 0.5 | 583.2 | 0.5 |
| | 400 | 317.0 | **4492.9** | 8464.9 | 0.5 | 5402.2 | 0.8 |
| C | 20 | 254.4 | 0.06 | 0.05 | 1.2 | **0.03** | 2.17 |
| | 30 | 1066.6 | 9.69 | 4.18 | 2.3 | **1.30** | 7.46 |
| | 40 | 2965.5 | 471.68 | 153.21 | 3.1 | **30.50** | 15.5 |
| D | 20 | 4087.7 | 23.6 | 10.9 | 2.2 | **1.9** | 12.5 |
| | 30 | 8834.5 | 8914.2 | 3625.3 | 2.5 | **1019.5** | 8.7 |

# Computational Experiments

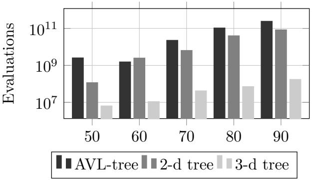
Average number of solution evaluations for 3-objective instances:



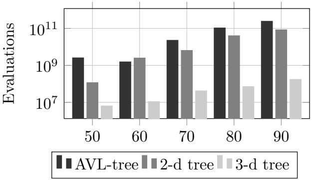

(a) Type A




(b) Type B




(c) Type C




(d) Type D

## Conclusions

The multi-dimensional indexing is applicable to the problem requiring considerably less solution evaluations, especially on hard instances.

## Conclusions

The multi-dimensional indexing is applicable to the problem requiring considerably less solution evaluations, especially on hard instances.

Algorithm speedup 2.3 for bi-dimensional cases and up to 15.5 on 3-dimensional cases.

# Conclusions

The multi-dimensional indexing is applicable to the problem requiring considerably less solution evaluations, especially on hard instances.

Algorithm speedup 2.3 for bi-dimensional cases and up to 15.5 on 3-dimensional cases.

The multi-dimensional indexing was not efficient on *easy* instances for which the set of solutions is relatively small.

# Conclusions

The multi-dimensional indexing is applicable to the problem requiring considerably less solution evaluations, especially on hard instances.

Algorithm speedup 2.3 for bi-dimensional cases and up to 15.5 on 3-dimensional cases.

The multi-dimensional indexing was not efficient on *easy* instances for which the set of solutions is relatively small.

Several instances are still intractable due the large number of intermediate solutions.

## Conclusions

The multi-dimensional indexing is applicable to the problem requiring considerably less solution evaluations, especially on hard instances.

Algorithm speedup 2.3 for bi-dimensional cases and up to 15.5 on 3-dimensional cases.

The multi-dimensional indexing was not efficient on *easy* instances for which the set of solutions is relatively small.

Several instances are still intractable due the large number of intermediate solutions.

For this reason this work proposes to use the indexing strategy in conjunction with an evolutionary metaheuristic.

# The Shuffled Complex Evolution

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

A population of $N * M$ individuals is randomly taken from the solution space;

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

A population of $N * M$ individuals is randomly taken from the solution space;

The population is then sorted by descending order of fitness and the best global solution is identified;

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

A population of $N * M$ individuals is randomly taken from the solution space;

The population is then sorted by descending order of fitness and the best global solution is identified;

The population is then shuffled into $N$ complexes, each containing $M$ individuals;

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

A population of $N * M$ individuals is randomly taken from the solution space;

The population is then sorted by descending order of fitness and the best global solution is identified;

The population is then shuffled into $N$ complexes, each containing $M$ individuals;

In this shuffling process the first individual goes to the first complex, the second individual goes to the second complex, individual $N$ goes to $N$-th complex, individual $(M + 1)$-th goes back to the first complex, etc;

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

A population of $N * M$ individuals is randomly taken from the solution space;

The population is then sorted by descending order of fitness and the best global solution is identified;

The population is then shuffled into $N$ complexes, each containing $M$ individuals;

In this shuffling process the first individual goes to the first complex, the second individual goes to the second complex, individual $N$ goes to $N$-th complex, individual $(M + 1)$-th goes back to the first complex, etc;

The next step after shuffling the complexes is to evolve each complex.

# The Shuffled Complex Evolution

The SCE regards a natural evolution happening simultaneously in independent communities;

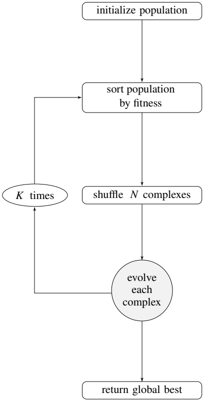
A population of $N * M$ individuals is randomly taken from the solution space;

The population is then sorted by descending order of fitness and the best global solution is identified;

The population is then shuffled into $N$ complexes, each containing $M$ individuals;

In this shuffling process the first individual goes to the first complex, the second individual goes to the second complex, individual $N$ goes to $N$-th complex, individual $(M+1)$-th goes back to the first complex, etc;

The next step after shuffling the complexes is to evolve each complex.



# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

At first the best individual of the subcomplex is considered for the crossing;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

At first the best individual of the subcomplex is considered for the crossing;

If the new solution is not better than the worst one, the best individual of the complex is considered for a crossing;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

At first the best individual of the subcomplex is considered for the crossing;

If the new solution is not better than the worst one, the best individual of the complex is considered for a crossing;

If the latter crossing did not result in any improvement, the best individual of whole population is considered;

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

At first the best individual of the subcomplex is considered for the crossing;

If the new solution is not better than the worst one, the best individual of the complex is considered for a crossing;

If the latter crossing did not result in any improvement, the best individual of whole population is considered;

Finally, if all the crossing steps couldn't generate a better individual, the worst individual of the subcomplex is replaced by a new random solution taken from the feasible solution space.

# The Shuffled Complex Evolution

In each step a subcomplex of $P$ individuals is selected from the complex;

After the selection of the subcomplex, its worst individual is identified to be replaced by a new generated solution;

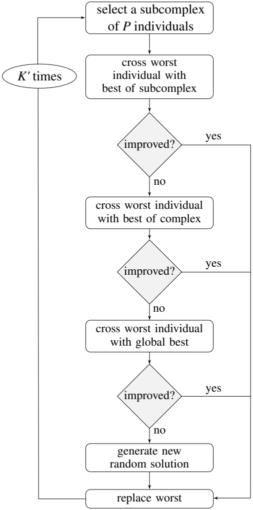
This new solution is generated by the crossing of the worst individual and an other individual with better fitness;

At first the best individual of the subcomplex is considered for the crossing;

If the new solution is not better than the worst one, the best individual of the complex is considered for a crossing;

If the latter crossing did not result in any improvement, the best individual of whole population is considered;

Finally, if all the crossing steps couldn't generate a better individual, the worst individual of the subcomplex is replaced by a new random solution taken from the feasible solution space.



# The Shuffled Complex Evolution

Use case for the MKP

# The SCE for MKP

the SCE is easily applied to any optimization problem. The only steps needed to be specified is **(a)** the creation of a new random solution and **(b)** the crossing procedure of two solutions.

# The SCE for MKP

New random solution generation for the MKP.

1: **function** NEW RANDOM SOLUTION

## The SCE for MKP

New random solution generation for the MKP.

```
function NEW RANDOM SOLUTION
    v ← shuffle(1, 2, . . . , n)
```

# The SCE for MKP

New random solution generation for the MKP.

**function** NEW RANDOM SOLUTION
    $v \leftarrow \text{shuffle}(1, 2, \ldots, n)$
    $s \leftarrow \emptyset$

# The SCE for MKP

New random solution generation for the MKP.

```
function NEW RANDOM SOLUTION
    v ← shuffle(1, 2, ..., n)
    s ← ∅
    for i ← 1 : n do
        s ← s ∪ {v_i}
```

# The SCE for MKP

New random solution generation for the MKP.

**function** NEW RANDOM SOLUTION
    $v \leftarrow \text{shuffle}(1, 2, \ldots, n)$
    $s \leftarrow \emptyset$
    **for** $i \leftarrow 1 : n$ **do**
        $s \leftarrow s \cup \{v_i\}$
        **if** $s$ is not feasible **then**

# The SCE for MKP

New random solution generation for the MKP.

```
function NEW RANDOM SOLUTION
    v ← shuffle(1, 2, ..., n)
    s ← ∅
    for i ← 1 : n do
        s ← s ∪ {v_i}
        if s is not feasible then
            s ← s − {v_i}
        end if
```

# The SCE for MKP

New random solution generation for the MKP.

```
function NEW RANDOM SOLUTION
    v ← shuffle(1, 2, ..., n)
    s ← ∅
    for i ← 1 : n do
        s ← s ∪ {v_i}
        if s is not feasible then
            s ← s − {v_i}
        end if
    end for
    return s
end function
```

## The SCE for MKP

Crossing procedure for the MKP.

1: **function** CROSSING($x^w$ : worst individual, $x^b$ : better individual, $c$)

# The SCE for MKP

Crossing procedure for the MKP.

1: **function** CROSSING($x^w$ : worst individual, $x^b$ : better individual, $c$)
2: $v \leftarrow \text{shuffle}(1, 2, \ldots, n)$

# The SCE for MKP

Crossing procedure for the MKP.

```
function CROSSING(x^w : worst individual, x^b : better individual, c)
    v ← shuffle(1, 2, ..., n)
    for i ← 1 : c  do
        j ← v_i
```

# The SCE for MKP

Crossing procedure for the MKP.

```
function CROSSING(x^w : worst individual, x^b : better individual, c)
    v ← shuffle(1, 2, ..., n)
    for i ← 1 : c do
        j ← v_i
        x_j^w ← x_j^b
    end for
```

# The SCE for MKP

Crossing procedure for the MKP.

**function** CROSSING($x^w$ : worst individual, $x^b$ : better individual, $c$)
    $v \leftarrow \text{shuffle}(1, 2, \ldots, n)$
    **for** $i \leftarrow 1 : c$ **do**
        $j \leftarrow v_i$
        $x^w_j \leftarrow x^b_j$
    **end for**
    **if** $s^w$ is not feasible **then**
        repair $s^w$
    **end if**

# The SCE for MKP

Crossing procedure for the MKP.

**function** CROSSING($x^w$ : worst individual, $x^b$ : better individual, $c$)
    $v \leftarrow$ shuffle$(1, 2, \ldots, n)$
    **for** $i \leftarrow 1 : c$ **do**
        $j \leftarrow v_i$
        $x_j^w \leftarrow x_j^b$
    **end for**
    **if** $s^w$ is not feasible **then**
        repair $s^w$
    **end if**
    return $s^w$
**end function**

## Computational Experiments

Parameters used for SCE:

- $N = 20$: number of complexes;
- $M = 20$: number of individuals in each complex;
- $P = 5$: number of individuals in each subcomplex;
- $K = 300$: number of algorithm iterations;
- $K' = 20$: number of iterations used in the complex evolving process;
- $c = n/5$: number of genes carried from parent in crossing process.

# Computational Experiments

SCE and SCEcr performance on Chu-Beasley problems.

| n | m | $\alpha$ | time (s) SCE | time (s) SCEcr | quality(%) SCE | quality(%) SCEcr |
|---|---|---|---|---|---|---|
| 100 | 5 | 0.25 | 1.22 | **0.17** | 96.51 | **99.73** |
| | | 0.50 | 1.34 | **0.18** | 97.42 | **99.86** |
| | | 0.75 | 1.37 | **0.17** | 98.87 | **99.91** |
| | 10 | 0.25 | 1.32 | **0.25** | 95.68 | **99.53** |
| | | 0.50 | 1.51 | **0.25** | 96.65 | **99.76** |
| | | 0.75 | 1.46 | **0.27** | 98.54 | **99.96** |
| | 30 | 0.25 | 1.74 | **1.20** | 95.38 | **97.96** |
| | | 0.50 | 1.79 | **0.89** | 96.41 | **99.18** |
| | | 0.75 | 1.72 | **0.95** | 98.18 | **99.52** |
| 250 | 5 | 0.25 | 2.87 | **0.69** | 93.22 | **99.86** |
| | | 0.50 | 2.82 | **0.70** | 94.88 | **99.94** |
| | | 0.75 | 2.93 | **0.69** | 97.57 | **99.96** |
| | 10 | 0.25 | 3.08 | **0.87** | 93.14 | **99.58** |
| | | 0.50 | 3.03 | **0.79** | 94.55 | **99.79** |
| | | 0.75 | 3.12 | **0.84** | 97.16 | **99.88** |
| | 30 | 0.25 | 3.74 | **1.52** | 93.10 | **98.42** |
| | | 0.50 | 3.74 | **1.36** | 94.20 | **99.33** |
| | | 0.75 | 3.99 | **1.48** | 96.64 | **99.59** |
| 500 | 5 | 0.25 | 5.62 | **1.25** | 91.37 | **99.77** |
| | | 0.50 | 5.72 | **1.24** | 93.39 | **99.88** |
| | | 0.75 | 5.88 | **1.20** | 96.42 | **99.92** |
| | 10 | 0.25 | 5.97 | **1.41** | 91.62 | **99.51** |
| | | 0.50 | 6.11 | **1.36** | 93.09 | **99.77** |
| | | 0.75 | 5.47 | **1.21** | 96.24 | **99.84** |
| | 30 | 0.25 | 6.20 | **1.96** | 91.37 | **98.76** |
| | | 0.50 | 6.26 | **1.82** | 92.56 | **99.42** |
| | | 0.75 | 6.05 | **1.73** | 95.97 | **99.67** |

# Computational Experiments

SCEcr performance on Glover-Kochenberger problems.

| # | n | m | time(s) SCE | time(s) SCEcr | quality(%) SCE | quality(%) SCEcr |
|---|---|---|---|---|---|---|
| 01 | 100 | 15 | 1.47 | **0.08** | 97.66 | **99.24** |
| 02 | 100 | 25 | 1.61 | **0.09** | 97.94 | **98.94** |
| 03 | 150 | 25 | 2.51 | **0.09** | 97.22 | **99.09** |
| 04 | 150 | 50 | 3.56 | **0.09** | 97.40 | **98.52** |
| 05 | 200 | 25 | 3.55 | **0.09** | 96.88 | **99.28** |
| 06 | 200 | 50 | 4.81 | **0.10** | 97.68 | **98.90** |
| 07 | 500 | 25 | 7.30 | **0.10** | 97.12 | **99.54** |
| 08 | 500 | 50 | 12.20 | **0.11** | 97.27 | **99.33** |
| 09 | 1500 | 25 | 24.61 | **0.12** | 95.40 | **98.22** |
| 10 | 1500 | 50 | 33.79 | **0.13** | 97.50 | **99.64** |
| 11 | 2500 | 100 | 121.28 | **0.15** | 97.95 | **99.70** |

## Conclusions

The SCE algorithm for MKP proved to be able to achieve fast convergence ratio, finding good quality near optimal solutions, demanding small amount of computational time.

## Conclusions

The SCE algorithm for MKP proved to be able to achieve fast convergence ratio, finding good quality near optimal solutions, demanding small amount of computational time.

The core concept as a variable fixing procedure proved to be efficient to reduce the size of the problems which provided fast execution time, producing higher quality solutions.

# Conclusions

The SCE algorithm for MKP proved to be able to achieve fast convergence ratio, finding good quality near optimal solutions, demanding small amount of computational time.

The core concept as a variable fixing procedure proved to be efficient to reduce the size of the problems which provided fast execution time, producing higher quality solutions.

At least 99.02% of best known, in less than 2 seconds for every instance.

# A SCE for the MOKP

# A SCE for the MOKP

As seen in previous sections the SCE is easily applied to any optimization problem.

## A SCE for the MOKP

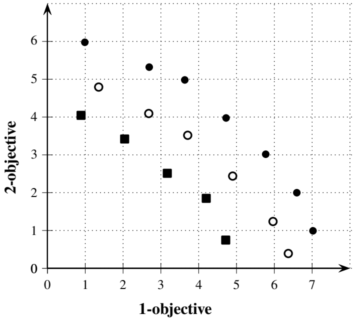
As seen in previous sections the SCE is easily applied to any optimization problem.

The fact that the MKP shares the same solution representation as the MOKP allow us to use the same procedures used on MKP.

# Non-dominated Sort

Fitness computation for multi-objective solutions.

**function** NDSORT($S$ : solution set)
    $i = 0$
    **while** $S \neq \emptyset$ **do**
        **for** $\boldsymbol{s} \in S$ **do**
            **if** $\nexists \boldsymbol{x} \in S : dom(\boldsymbol{x}, \boldsymbol{s})$ **then**
                $o_s = i$
                $S = S - x$
            **end if**
        **end for**
        $i = i + 1$
    **end while**
**end function**



# Final Doctoral Schedule

| Activities | Weeks | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | November | | | December | | | | January | | | |
| | 2° | 3° | 4° | 1° | 2° | 3° | 4° | 1° | 2° | 3° | 4° |
| Literature review | ● | ● | ● | | | | | | | | |
| Implementation and adjustments | ● | ● | ● | ● | | | | | | | |
| Computational experiments | | | ● | ● | ● | | | | | | |
| CEC Paper writing | | | | | | ● | ● | ● | | | |
| CEC Paper submission | | | | | | | | | ● | | |
| Thesis writing | | | | ● | ● | ● | ● | ● | ● | ● | |