Stochastic Local Search and Metaheuristics

S. Loudni

GREYC, CNRS UMR 6072 - University of Caen - France

Master 2 DECIM – UE métaheuristiques – 2016/2017

Une instance d'un problème d'optimisation combinatoire est définie par un triplet $(\mathcal{S}_p, \mathcal{S}_a, f)$ tel que :

- $\mathcal{S}_p$ est l'ensemble des solutions potentielles ;
- $\mathcal{S}_a \subseteq \mathcal{S}_p$ est l'ensemble des solutions admissibles ;
- f est une fonction de $\mathcal{S}_p$ dans $\boldsymbol{R}$.

Le problème est de trouver un élément $s^* \in \mathcal{S}_a$ qui minimise (ou maximise) la valeur de la fonction coût f.

➥ L'élément s^* s'appelle un optimum global.

Neighborhoods (1/2)

Consider an instance $\mathcal{I} = (\mathcal{S}, f)$ of an optimization problem:

- A neighborhood function is a mapping $N_{\mathcal{I}} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, where $2^{\mathcal{S}}$ denotes the powerset $\{V \mid V \in \mathcal{S}\}$.
- The neighborhood function specifies for each solution $s \in \mathcal{S}$ a set $N_{\mathcal{I}}(s) \subseteq \mathcal{S}$, which is called the neighborhood of s.
  ➥ set of points that are "close" in some sense to the point s.
- The cardinality of $N_{\mathcal{I}}(s)$ is called the neighborhood size of s.

Neighborhoods (2/2)

A neighborhood function $N_{\mathcal{I}}$ is also defined through an operator.

➥ An operator Δ is a collection of operator functions $\delta : \mathcal{S} \to \mathcal{S}$ such that :

$$\boxed{s' \in N_{\mathcal{I}}(s) \iff \exists \delta \in \Delta \mid \delta(s) = s'}$$

The operator function δ is called a move, and the application of δ to s can be noted as $s' = s \oplus \delta$

➥ One can express the neighborhood in terms of moves :

$$\boxed{N_{\mathcal{I}}(s) = \{s' \mid s' = s \oplus \delta, \delta \in \Delta\}}$$

A given neighborhood $N_{\mathcal{I}}(.)$ for an instance $\mathcal{I}$ of an optimization problem induces a neighborhood graph $G_{\mathcal{I}} = (V_{\mathcal{I}}, A_{\mathcal{I}})$.

- The vertex set $V_{\mathcal{I}}$ corresponds to the set $\mathcal{S}$ of solutions.
- The arc set $A_{\mathcal{I}}$ is defined such that $(i, j) \in A_{\mathcal{I}}$ iff $j \in N_{\mathcal{I}}(i)$.

➥ The neighborhood graph can be considered as an undirected graph if the neighborhood is symmetric, i.e.

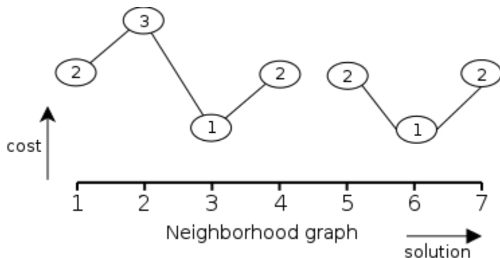
$$\boxed{\text{if } s' \in N_{\mathcal{I}}(s) \Longleftrightarrow s \in N_{\mathcal{I}}(s') \text{ for all } s, s' \in V_{\mathcal{I}}}$$

Global vs. Local Optima

- A solution $\hat{s}$ is called locally optimal with respect to $N_{\mathcal{I}}$ if $f(\hat{s}) \leq f(s)$ for all $s \in N_{\mathcal{I}}(\hat{s})$ with $(\hat{s},s) \in A_{\mathcal{I}}$.
- A solution s^* is called globally optimal if $f(s^*) \leq f(s)$ for all $s \in \mathcal{S}$.

Neighborhood graphs: an example

solution	$f(s)$	$N(s)$
1	2	$\{2\}$
2	3	$\{1,3\}$
3	1	$\{2,4\}$
4	2	$\{3\}$
5	2	$\{6\}$
6	1	$\{5,7\}$
7	2	$\{6\}$



- ➥ Two global optima, namely solutions 3 and 6
- ➥ The neighborhood graph consists of two disconnected components (the solution space is partitioned into two subsets)

Iterative improvement Scheme

If terminating, the following general algorithmic scheme obviously returns a local minimum:

- **Step 1**: Start with an arbitrary feasible solution $s \in \mathcal{S}$.
- **Step 2**: While there is some $s' \in N(s)$ with $f(s') < f(s)$:
  a) Select one such s'.
  b) s' becomes the new s (i.e., $s = s'$).
- **Step 3**: Return s as the final solution.

➥ Different heuristics for selecting a neighboring solution: **first** and **best improvement**.

Algorithme 1: Steepest Descent.

```
function BestImprovement(s) ;
begin
1    repeat
2        choose s' ∈ N(s) s.t. f(s') = min_{x∈N(s)} f(x) ;
3        if f(s') < f(s) then
4            s ← s';
         end
     until s is a Local optimum;
end
```

➥ At each step, the neighbourhood $N(s)$ of s is explored completely, which may be time-consuming.

First Improvement Heuristic

Algorithme 2: First Descent.

```
function FirstImprovement(s) ;
begin
1    repeat
2        choose s_i ∈ N(s) using a predefined random ordering ;
3        if f(s_i) < f(s) then
4            s ← s_i;
         end
     until s is Local optimum;
end
```

➥ Neighboring solutions $s_i \in N(s)$ are enumerated systematically and a move is made as soon as a direction for the descent is found.

LS components

- **Search Space**: defined by the solution representation.
- **Neighborhood function**
- **Evaluation function**
- **Search step (or move)**
  pair of search positions s, s' for which s' can be reached from s in one step.
  - **Search trajectory:** finite sequence of search positions $< s_0, s_1, \ldots, s_k >$ such that (s_{i-1}, s_i) is a search step for any $i \in \{1, \ldots, k\}$.

Example (1/2)

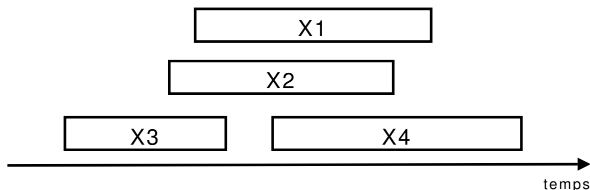
Allocation de ressources à des tâches:

- Étant donné un certain nombre de tâches (T_1 à T_4), dont chacune peut être exécutée par une ressource parmi un ensemble restreint.
- Objectif: trouver une assignation des ressources aux tâches de manière à ce qu'aucune ressource n'effectue plus d'une tâche en même temps.

Modélisation CSP:

- **Variables** $X_1..X_n$ correspondant aux tâches $T_1, T_2..T_n$.
- **Domaines** = ressources qui peuvent être effectuées par la tâche.
- **Contraintes** = deux tâches se chevauchant dans le temps ne peuvent être effectuées par la même ressource.

Example (2/2)



X1 = {B,C}
X2 = {A,C}
X3 = {B,C}
X4 = {A,B}

Les contraintes:

C(x1,x2) : {(B,A), (B,C), (C,A)}
C(x1,x3) : {(B,C), (C,B)}
C(x1,x4) : {(B,A), (C,B), (C,A)}
C(x2,x3) : {(A,B), (A,C), (C,B)}
C(x2,x4) : {(A,B), (C,A), (C,B)}

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$		

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$		

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$		

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_4 \rightarrow \text{B}$		

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_4 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_1, x_4)$	2

Min-conflict heuristic

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_4 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_1, x_4)$	2

Accept $(x_1 \rightarrow \text{C})$: $s' = (x_1 = \text{C}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow$ B		

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \to \text{C}$		

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \to \text{C}$	$c(x_1, x_2)$	1

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \to \text{C}$	$c(x_1, x_2)$	1
$x_3 \to \text{C}$		

Min-conflict heuristic

second iteration: $s = (x_1 = \mathrm{C},\ x_2 = \mathrm{A},\ x_3 = \mathrm{B},\ x_4 = \mathrm{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \mathrm{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \to \mathrm{C}$	$c(x_1, x_2)$	1
$x_3 \to \mathrm{C}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2

Min-conflict heuristic

second iteration: $s = (x_1 = \mathrm{C},\ x_2 = \mathrm{A},\ x_3 = \mathrm{B},\ x_4 = \mathrm{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \mathrm{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \rightarrow \mathrm{C}$	$c(x_1, x_2)$	1
$x_3 \rightarrow \mathrm{C}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_4 \rightarrow \mathrm{B}$		

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \rightarrow \text{C}$	$c(x_1, x_2)$	1
$x_3 \rightarrow \text{C}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_4 \rightarrow \text{B}$	–	0

Min-conflict heuristic

second iteration: $s = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_2 \to \text{C}$	$c(x_1, x_2)$	1
$x_3 \to \text{C}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_4 \to \text{B}$	–	0

Accept $(x_4 \to \text{B})$: $s' = (x_1 = \text{C},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{B})$
$\to$ Solution!!

Min-conflict heuristic - local optimum

first iteration: $s = (x_1 = \text{B}, x_2 = \text{A}, x_3 = \text{B}, x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_4 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_1, x_4)$	2

Min-conflict heuristic - local optimum

first iteration: $s = (x_1 = \text{B},\ x_2 = \text{A},\ x_3 = \text{B},\ x_4 = \text{A})$

$\rightarrow$ 2 **conflicts**: $c(x_1, x_3)$ et $c(x_2, x_4)$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_2 \rightarrow \text{C}$	$c(x_1, x_3)$	1
$x_3 \rightarrow \text{C}$	$c(x_2, x_4)$	1
$x_4 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_1, x_4)$	2

Accept $(x_2 \rightarrow \text{C})$: $s' = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B}, x_2 = \text{C}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \mathrm{B},\ x_2 = \mathrm{C},\ x_3 = \mathrm{B},\ x_4 = \mathrm{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \mathrm{C}$		

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B}, x_2 = \text{C}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{C}$	$c(x_1, x_2)$	1

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{C}$	$c(x_1, x_2)$	1
$x_2 \to \text{A}$		

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_1, x_2)$	1
$x_2 \rightarrow \text{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{C}$	$c(x_1, x_2)$	1
$x_2 \to \text{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_3 \to \text{C}$		

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B},\ x_2 = \text{C},\ x_3 = \text{B},\ x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{C}$	$c(x_1, x_2)$	1
$x_2 \to \text{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_3 \to \text{C}$	$c(x_2, x_3)$	1

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B}, x_2 = \text{C}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \to \text{C}$	$c(x_1, x_2)$	1
$x_2 \to \text{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_3 \to \text{C}$	$c(x_2, x_3)$	1
$x_4 \to \text{B}$		

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \mathrm{B},\ x_2 = \mathrm{C},\ x_3 = \mathrm{B},\ x_4 = \mathrm{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \mathrm{C}$	$c(x_1, x_2)$	1
$x_2 \rightarrow \mathrm{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_3 \rightarrow \mathrm{C}$	$c(x_2, x_3)$	1
$x_4 \rightarrow \mathrm{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2

Min-conflict heuristic - local optimum

second iteration: $s = (x_1 = \text{B}, x_2 = \text{C}, x_3 = \text{B}, x_4 = \text{A})$

modified variable	conflicts with	total number of conflicts
$x_1 \rightarrow \text{C}$	$c(x_1, x_2)$	1
$x_2 \rightarrow \text{A}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2
$x_3 \rightarrow \text{C}$	$c(x_2, x_3)$	1
$x_4 \rightarrow \text{B}$	$c(x_1, x_3)$ and $c(x_2, x_4)$	2

No change that decreases the number of conflicts

$\rightarrow$ STOP

The choice of neighborhood is critical part for the effectiveness of any local search method.

We will look at some examples for TSP and Graph Coloring Problem.

Solution Representations and Neighborhoods

Many combinatorial optimization problems can be naturally formulated as a search for a **permutation** or an **assignment** of n elements.

Permutation:

- Assignment problems (quadratic assignment)
- Traveling Salesman Problem (TSP)

Assignment:

- Graph Coloring
- CSP, SAT

Permutations

$\Pi(n)$ indicates the set all permutations of the numbers $\{1, 2, \ldots, n\}$

$(1, 2, \ldots, n)$ is the identity permutation

if $\pi \in \Pi$ and $1 \leq i \leq n$ then:

- π_i is the element at position i
- $pos_\pi(i)$ is the position of element i

Neighborhood Operators for Permutations

Swap operator interchanges two successive positions in a permutation

$$\Delta_S = \{\delta_S^i \mid 1 \leq i \leq n\}$$

$$\delta_S^i(\pi_1 \ldots \pi_i \pi_{i+1} \ldots \pi_n) = (\pi_1 \ldots \pi_{i+1} \pi_i \ldots \pi_n)$$

2-exchange operator switches two different positions in a permutation

$$\Delta_X = \{\delta_X^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_X^{ij}(\pi) = (\pi_1 \ldots \pi_{i-1} \pi_j \pi_{i+1} \ldots \pi_{j-1} \pi_i \pi_{j+1} \ldots \pi_n)$$

Insertion operator deletes one element from the permutation, and inserts it in a new position

$$\Delta_I = \{\delta_X^{ij} \mid 1 \leq i \leq n, 1 \leq j \leq n, i \neq j\}$$

$$\delta_I^{ij}(\pi) = \begin{cases} (\pi_1 \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_n) & i < j \\ (\pi_1 \ldots \pi_j \pi_i \pi_{j+1} \ldots \pi_{i-1} \pi_{i+1} \ldots \pi_n) & i > j \end{cases}$$

Neighborhood Operators for Assignments

An assignment can be represented as a mapping
$\sigma : \{X_1 \dots X_n\} \rightarrow \{(d_1 \dots d_n) \mid d_i \in D(X_i), i \in \{1, \dots, n\}\}$

$$\sigma = \{X_1 = d_1, \dots X_n = d_n\}$$

1-Flip operator

$$\Delta_{1F} = \{\delta_{1F}^{ik} \mid 1 \leq i \leq n, 1 \leq k \leq \mid D(X_i) \mid\}$$

$$\delta_{1F}^{ik}(\sigma') = \{\sigma' : \sigma'[X_i] = d_k \text{ and } \sigma'[X_j] = \sigma[X_j] \ \forall i \neq j\}$$

2-exchange operator

$$\Delta_{2E} = \{\delta_{2E}^{ij} \mid 1 \leq i < j \leq n\}$$

$$\delta_{2E}^{ij}(\sigma') = \{\sigma' : \sigma'[X_i] = \sigma[X_j], \sigma'[X_j] = \sigma[X_i] \text{ and } \sigma'[X_k] = \sigma[X_k] \ \forall k \neq i, j\}$$

Example I: The Travelling Salesman Problem (TSP) (1/2)

- **Problem Statement**:
  - **Given**: Directed, arc-weighted graph G corresponding to n cities denoted by $C = \{1, 2, \ldots n\}$, where the weight of an arc (i, j) is given by the distance $d_{ij} \in \boldsymbol{N}^+$ from city i to city j.
  - **Objective**: Find a tour with minimum total tour length, i.e., find a permutation π of C that minimizes

$$\sum_{i=1}^{n-1} d_{\pi_i,\pi_{i+1}} + d_{\pi_n,\pi_1}$$

  - permutation π specifies the tour in which $\pi(i)$ is the ith city that is visited.

➥ TSP can be viewed as finding a **minimal-weighted Hamiltonian cycle** in G.

Example I: The Travelling Salesman Problem (TSP) (2/2)

Types of TSP instances:

- **Symmetric**: for each pair $i, j \in C$, $d_{ij} = d_{ji}$
- **Euclidien**: cities are points in a Euclidean space, weight function is the Euclidean distance.

Why is the TSP difficult ?

- **TSP is NP-hard**: the size of the solution space is given by the number of permutations of C: n!
- **25 cities**: $25! = 15,511,210,043,330,985,984,000,000$

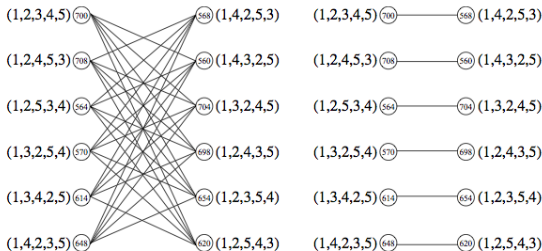
TSP : The five western islands of the Canary Islands

$$d = \begin{pmatrix} 0 & 233 & 152 & 118 & 227 \\ 223 & 0 & 95 & 115 & 90 \\ 152 & 95 & 0 & 30 & 93 \\ 118 & 115 & 30 & 0 & 115 \\ 227 & 90 & 93 & 115 & 0 \end{pmatrix}$$

Find the shortest tour that starts and ends in Maspalomas and that visits each of four cities exactly once.

- The five cities are all reachable by boat, and the distances in kilometers between the cities are given by the symmetric distance matrix.

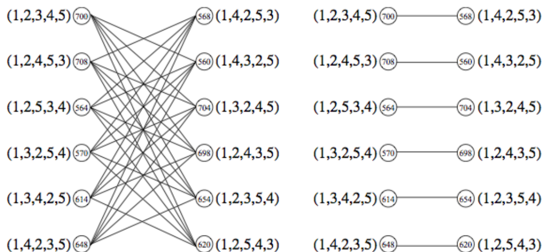
The Neighborhood graph for the 2-Exchange operator



The neighborhood graph is a complete bipartite graph, except edges on the right figure:

- We leave loops out of consideration,
- We only consider tours in which the cities 1, 2, and 5 are visited in increasing order.

The Neighborhood graph for the 2-Exchange operator



- The diameter of the neighborhood graph is three.
- The optimal tour has a cost of 560 and is given by $(1, 4, 3, 2, 5)$

Approximate Solutions for TSP

Two types:

- **Construction heuristics**
  ➥ A tour is built from nothing.

- **Improvement heuristics**
  ➥ Start with 'some' tour, and continue to change it into a better one as long as possible.

Construction heuristics for TSP (1/2)

Nearest Neighbour
This is the simplest and most straight forward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.
➥ Time Complexity: $O(n^2)$

Construction heuristics for TSP (1/2)

Nearest Neighbour

This is the simplest and most straight forward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

➥ Time Complexity: $O(n^2)$

1. Select a random city.

Construction heuristics for TSP (1/2)

Nearest Neighbour

This is the simplest and most straight forward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

➥ Time Complexity: $O(n^2)$

1. Select a random city.
2. Find the nearest unvisited city and go there.

Construction heuristics for TSP (1/2)

Nearest Neighbour

This is the simplest and most straight forward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

➥ Time Complexity: $O(n^2)$

1. Select a random city.
2. Find the nearest unvisited city and go there.
3. Are there any unvisitied cities left? If yes, repeat step 2.

Construction heuristics for TSP (1/2)

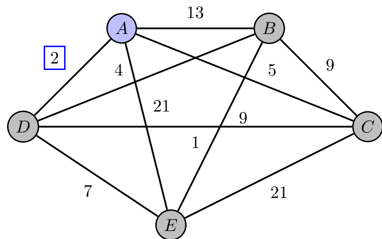
Nearest Neighbour

This is the simplest and most straight forward TSP heuristic. The key to this algorithm is to always visit the nearest city, then return to the starting city when all the other cities are visited.

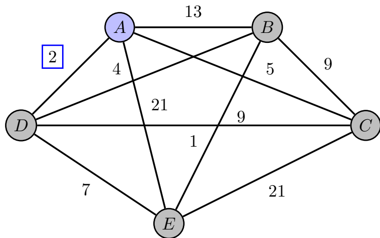
➥ Time Complexity: $O(n^2)$

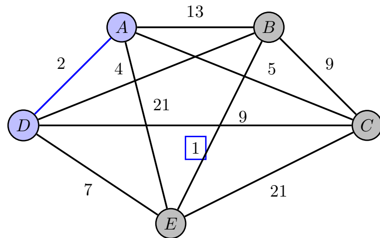
1. Select a random city.
2. Find the nearest unvisited city and go there.
3. Are there any unvisitied cities left? If yes, repeat step 2.
4. Return to the first city.

Nearest Neighbor Heuristic: An Example (1/2)

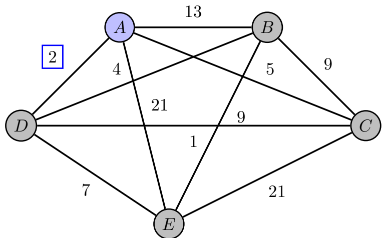


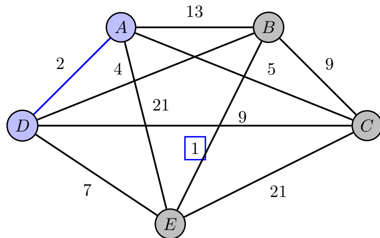
Nearest Neighbor Heuristic: An Example (1/2)

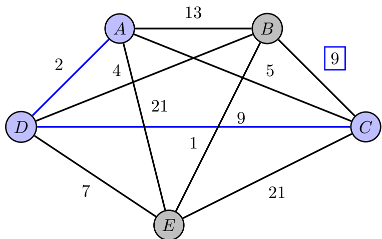




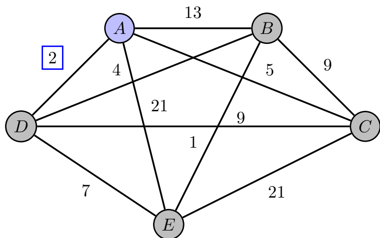
Nearest Neighbor Heuristic: An Example (1/2)

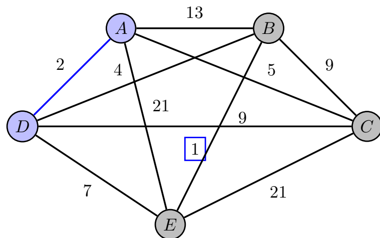


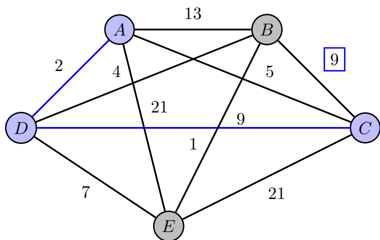


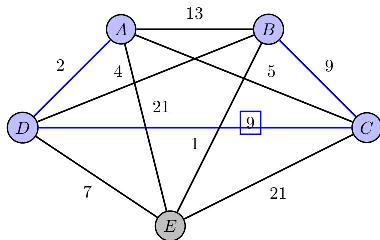


Nearest Neighbor Heuristic: An Example (1/2)

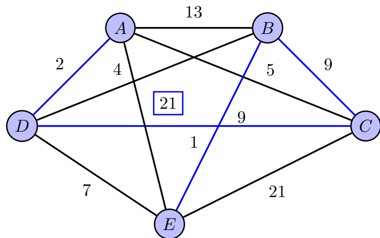




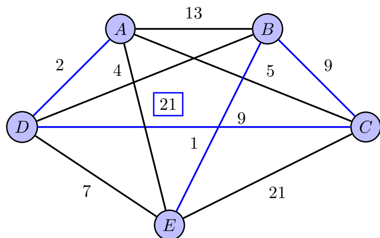


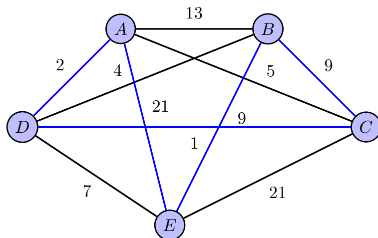


Nearest Neighbor Heuristic: An Example (2/2)

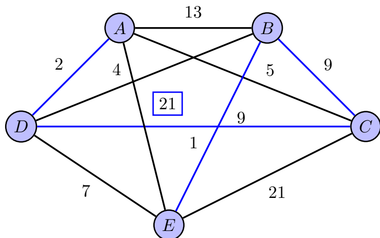


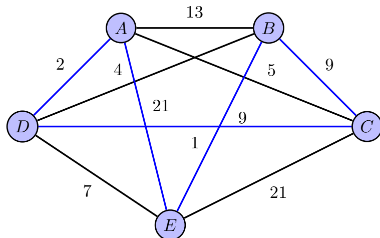
Nearest Neighbor Heuristic: An Example (2/2)

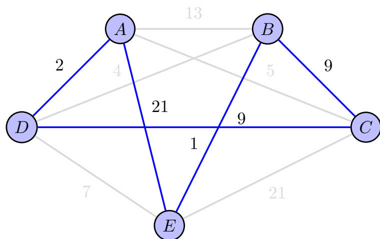




Nearest Neighbor Heuristic: An Example (2/2)







The total distance of the path $A \rightarrow D \rightarrow C \rightarrow B \rightarrow E \rightarrow A$ obtained using the nearest NNH is $2 + 1 + 9 + 9 + 21 = 42$.

Construction heuristics (2/2)

Greedy heuristic:

1. Starts with the departure Node 1.
2. Choose the cheapest edge and put it in the tour.
3. Add the cheapest edge not already in the tour, which if added, would not create a subtour, to the tour.
   Also the edge to be added can't create a degree 3 node!
4. The process continues until all the nodes are visited once and only once then back to Node 1.

When the algorithm is terminated, the sequence is returned as the best tour.

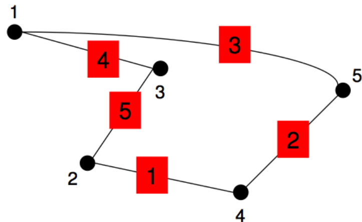
➥ Time Complexity: $O(n^2 log_2(n))$

Greedy heuristic: An Example

	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2

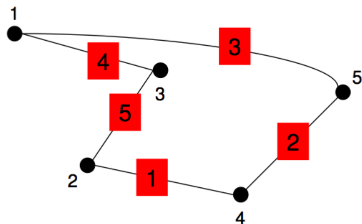
Greedy heuristic: An Example

	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2



Greedy heuristic: An Example

	2	3	4	5
1	2.9	7.4	7.1	2.1
2		7.7	2	9.6
3			6.7	3.9
4				2



Cost of the tour is
$2 + 2 + 2.1 + 7.4 + 7.7 = 21.2.$

Start with a tour (e.g., from heuristic) and improve it stepwise

- 2-OPT
- 3-OPT
- k-OPT
- Lin-Kernighan
- ...

k-OPT heuristic

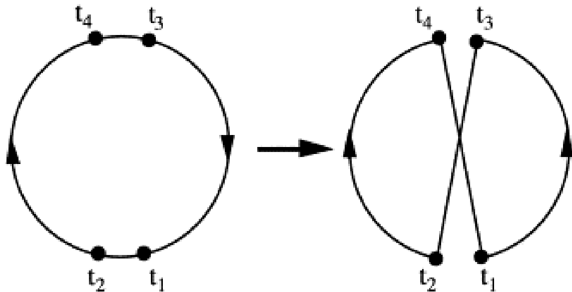
This is an iterative improvement heuristic based on the k-exchange operator :

- A fixed number $k \geq 2$ of arcs is removed, and k appropriate arcs are introduced to re-connect the partial tours.
- The number of operations to test all k-exchanges increases rapidly as the number of cities increases.
- In a naive implementation the testing of a k-exchange has a time complexity of $O(n^k)$.
- In practice, only 2-OPT (or 3-OPT) are usually applied.

2-OPT move

- Replace two non-adjacent edges (t_i, t_{i+}) and (t_j, t_{j+}) by two others (t_i, t_j) and (t_{i+}, t_{j+}).
  ➥ They represent the only two edges that can create a valid tour.

- To maintain a consistent orientation of the tour by the predecessor-successor relationship, one of the two subpaths remaining after dropping the first two edges must be reversed, namely the subpath $(t_{i+}, \ldots, t_j)$.



2 arcs of the current tour are replaced by 2 new arcs in such a way that a shorter tour is achieved.

3-OPT move

Consists of deleting three edges of the current tour (instead of two) and relinking the endpoints of the resulting subpaths in the best possible way.

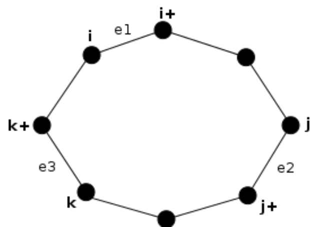
For example, letting (t_i, t_{i+}), (t_j, t_{j+}) and (t_k, t_{k+}) denote the triplet of edges deleted from the current tour. At least three possible ways to relink the three subpaths consist of:

(a) creating edges (t_i, t_{j+}), (t_k, t_{i+}) and (t_j, t_{k+})

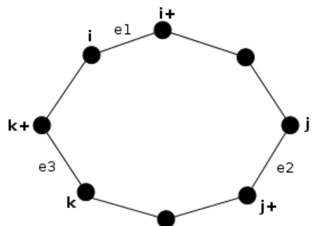
(b) creating edges (t_i, t_j), (t_{j+}, t_{k+}) and (t_{i+}, t_k)

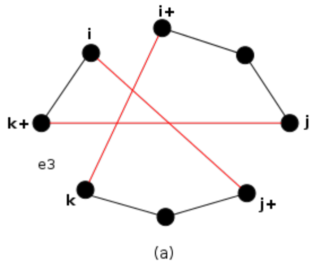
(c) creating edges (t_{i+}, t_{j+}), (t_j, t_{k+}) and (t_i, t_k)

3-OPT move (con't)



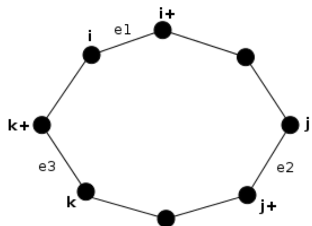
3-OPT move (con't)

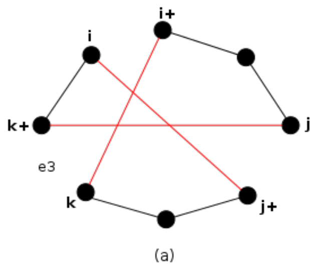




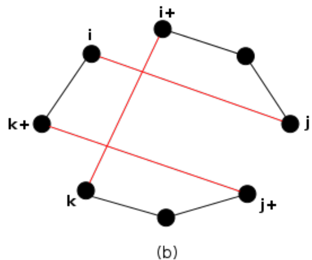
(a)

3-OPT move (con't)



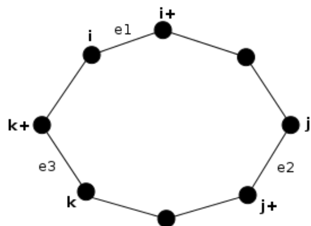


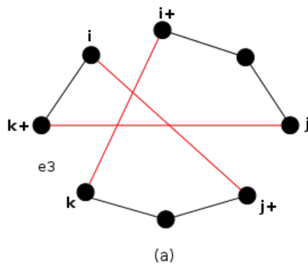
(a)



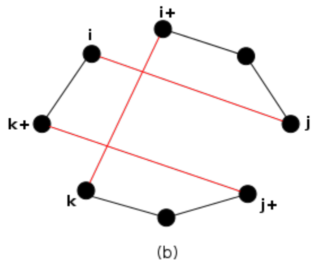
(b)

3-OPT move (con't)

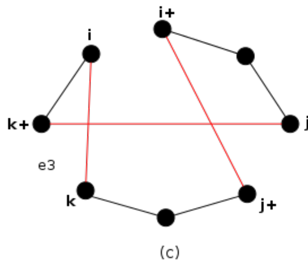




(a)



(b)

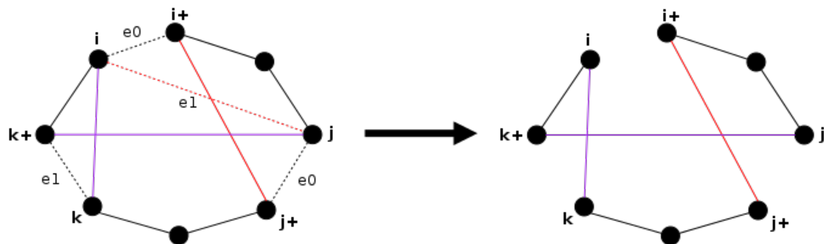


(c)

3-OPT move (con't)

An important difference between the two possibilities (a) and (b) to create 3-OPT moves is that the orientation of the tour is preserved in (a), while in (b) the subpaths $(t_{i+}, \ldots, t_j)$, and (t_{j+}, t_k) have to be reversed to maintain a feasible tour orientation.

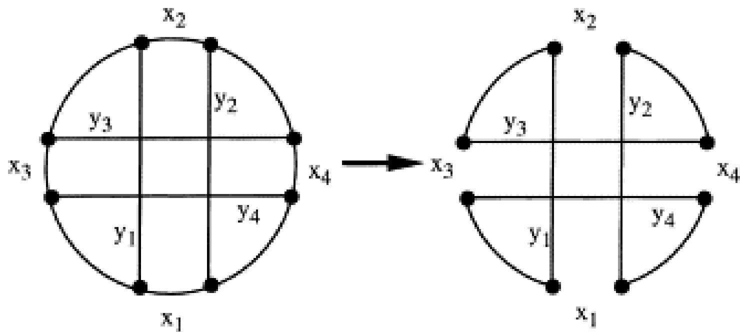
Class of 3-OPT moves



3-OPT move obtained as a sequence of two (non-independent) 2-OPT moves.

➥ This special case may occur when an edge inserted by the first 2-OPT move is deleted by the application of the second 2-OPT move.

Example of Double-Bridge move



Incremental updates

- **Key idea**: calculate effects of differences between current search position s and neighbours s' on evaluation function value.
- Evaluation function values often consist of independent contributions of solution components; hence, $f(s')$ can be efficiently calculated from $f(s)$ by differences between s and s' in terms of solution components.
- Typically crucial for the efficient implementation of II algorithms.

Example: Incremental updates for Symmetric TSP

- **Solution components** = edges of given graph G

- We know the length ℓ of a tour π and we want to know the length ℓ' of a neighboring tour π' of π.
  - ➥ computing ℓ' from scratch would take $O(n)$.
  - ➥ with incremental updates, it can be done in constant time.

$$\ell' = \ell - \sum_{i=1}^{k'} d(e_i) + \sum_{i=1}^{k'} d(e_i')$$

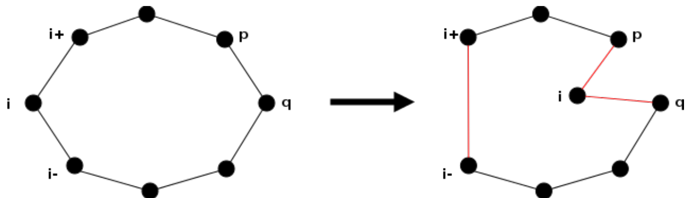
where $d(e)$ denotes d_{ij} for an edge $e = \{i, j\}$ and where the edges $e_1 \ldots, e_{k'}$ with $k' \leq k$ are removed and the edges $e_1', \ldots, e_{k'}'$ are added by the k-exchange that turns π into π'.

- For the 2-exchange neighborhood, an iteration of iterative improvement can be implemented to run in $O(n^2)$ time.

TSP: Node Insertion moves

Two types of node-based moves:

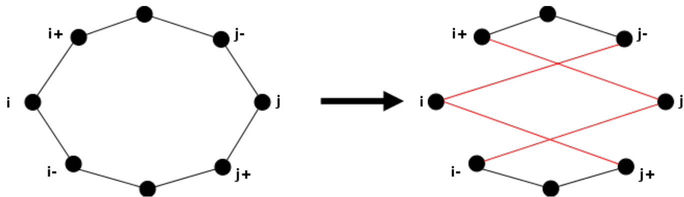
(1) node insertion moves: a selected node t_i is inserted between two adjacent nodes t_p and t_q in the tour by adding edges (t_p, t_i), (t_i, t_q) and (t_{i-}, t_{i+}) and dropping edges (t_{i-}, t_i), (t_i, t_{i+}) and (t_p, t_q).



TSP: Node Insertion moves

Two types of node-based moves:

(2) node exchange moves: two nodes t_i and t_j exchange positions by adding edges (t_{i-}, t_j), (t_j, t_{i+}), (t_{j-}, t_i), (t_i, t_{j+}) and dropping edges (t_{i-}, t_i), (t_i, t_{i+}), (t_{j-}, t_j), (t_j, t_{j+}).



OR-OPT heuristic

The Or-opt heuristic proceeds as a multi-stage generalized insertion process:

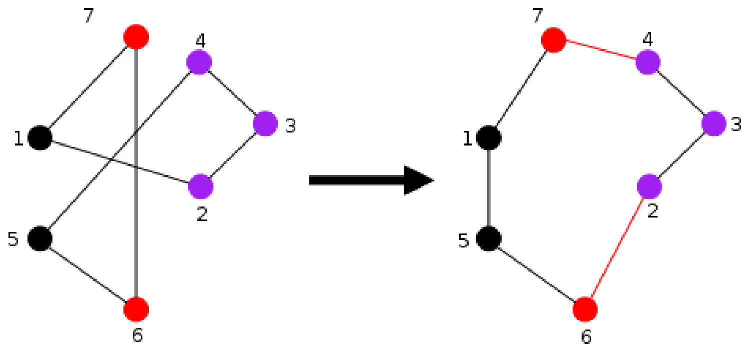
- Start by considering the insertion of three-node subpaths (between two adjacent nodes).
- Then successively reduce the process to insert two-node subpaths (hence edges) and finally to insert single nodes.
- Change the type of move employed whenever a local optimum is found for the current neighborhood.

OR-OPT heuristic (con't)

Special case of 3-OPT in which sub-paths of 3 adjacent nodes in the tour are inserted between two other nodes.

OR-OPT heuristic (con't)

Special case of 3-OPT in which sub-paths of 3 adjacent nodes in the tour are inserted between two other nodes.

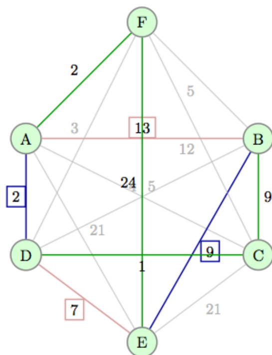


2-OPT heuristic : An Example

A naive implementation of 2-opt runs in $O(n^2)$, this involves selecting an edge (t_1,t_2) and searching for another edge (t_3,t_4), completing a move only if :

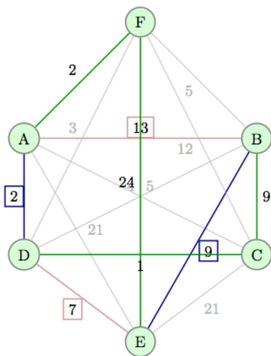
$$dist(t_1, t_2) + dist(t_3, t_4) > dist(t_2, t_3) + dist(t_1, t_4)$$

The total distance of the tour is 47



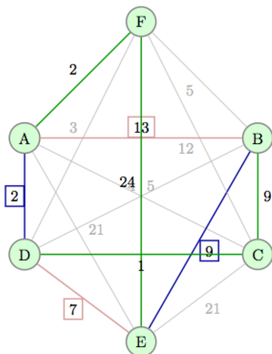
2-OPT heuristic : An Example (con't)

The total distance of the tour is 47



Let AD be the first edge being considered. We select an edge such that it is not adjacent to AD. Here, we only one such edge, BE. We can replace AD and BE with AB and DE.

The total distance of the tour is 47

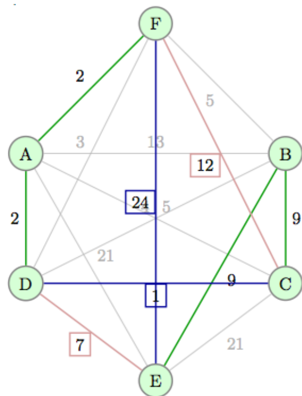


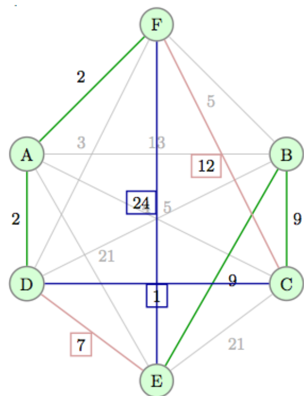
Let AD be the first edge being considered. We select an edge such that it is not adjacent to AD. Here, we only one such edge, BE. We can replace AD and BE with AB and DE.

➥ Here we can see that that the sum of replaceable edges is higher than the sum of the original edges. So, we do not replace the edges.

2-OPT heuristic : An Example (con't)

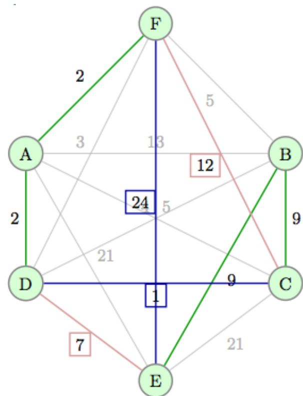
We consider the next edge, DC. Its has only one non adjacent edge FE.

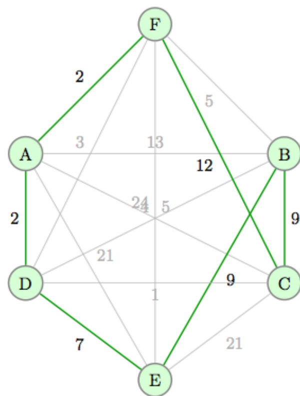




➥ Here, as we have the sum of current edges greater than the sum of replaceable edges, we replace them with the replaceable edges.

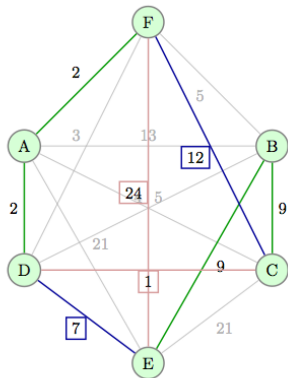
2-OPT heuristic : An Example (con't)



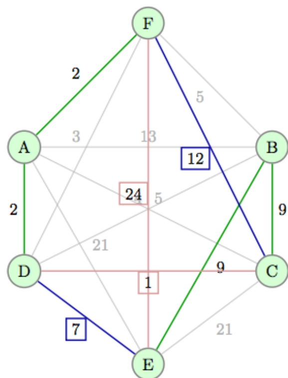


➥ Here, as we have the sum of current edges greater than the sum of replaceable edges, we replace them with the replaceable edges.

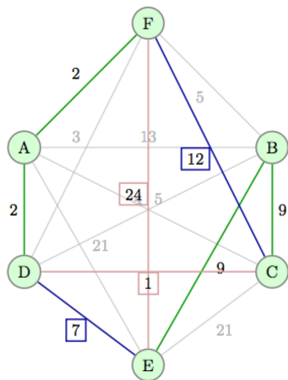
We now consider the edge DE. Its has only one non adjacent edge CF.



2-OPT heuristic : An Example (con't)



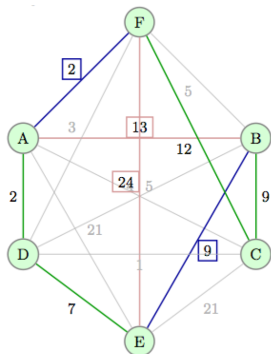
➥ As the replaceable edges are larger than the original edges, we don't replace the original edges.



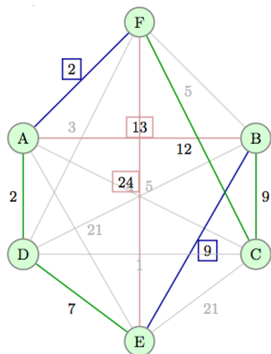
We go to the next step where the edge EB is selected. The edge that is non adjacent to this edge is FA.

➥ As the replaceable edges are larger than the original edges, we don't replace the original edges.

2-OPT heuristic : An Example (con't)

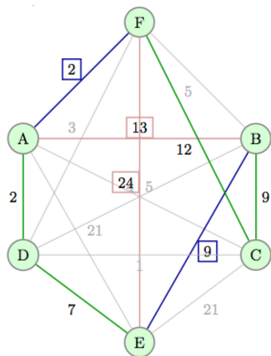


2-OPT heuristic : An Example (con't)



No edge replacement is necessary in this case.

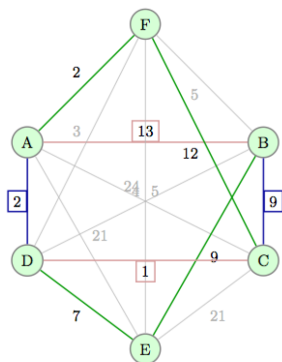
2-OPT heuristic : An Example (con't)

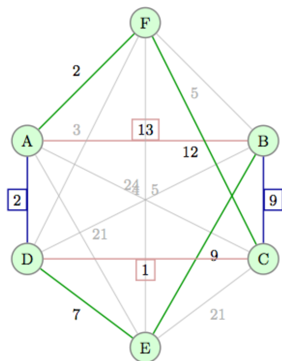


No edge replacement is necessary in this case.

➥ We now move on to the next edge, BC. The non adjacent edge to this edge is AD.

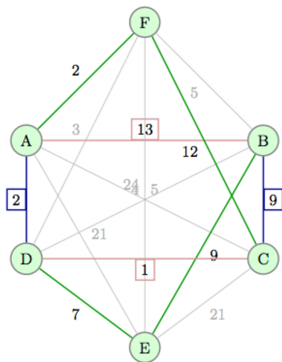
2-OPT heuristic : An Example (con't)





No edge replacement is necessary in this case.

2-OPT heuristic : An Example (con't)



No edge replacement is necessary in this case.

➥ Now we consider the next edge CF. The non adjacent edge of this edge is DE.

➥ ...

Example II: The Graph Coloring Problem (GCP) (1/2)

A (vertex) coloring of a graph $G = (V, E)$ is a mapping $c : V \rightarrow C$.

- The elements of C are called colors; the vertices of one color form a color class.
- If $|C| = k$, we say that c is a k-coloring (often we use $C = \{1, \ldots, k\}$).
- A coloring is legal if adjacent vertices have different colors.
- A graph is k-colorable if it has a legal k-coloring.

➥ In a legal coloring, each color class is a **stable set**. Hence a k-coloring may also be seen as a **partition** of the vertex set of G into k **disjoint stable sets** $V_i = \{v | c(v) = i\}$ for $1 \leq i \leq k$.

Example II: The Graph Coloring Problem (GCP) (2/2)

The k-coloring decision problem : Given graph $G = (V, E)$ and an integer k, determine if G can be colored with k colors with no edge with both ends of the same color.

The k-coloring optimization problem : Find a k-coloring that minimizes the number of edges with both ends of the same color (conflicts).

- The number of conflicts is our objective function.

➥ Complexity : NP-Complet for $k \geq 3$

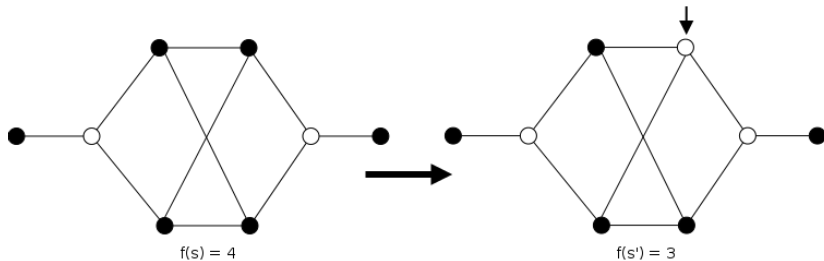
Defining Neighborhood: GCP

1-Flip move

It consists of changing the original color class $V_{c(v)}$ of a single conflicting vertex v to its best possible new color class V_i, with $c(v) \neq i$.

1-Flip move

It consists of changing the original color class $V_{c(v)}$ of a single conflicting vertex v to its best possible new color class V_i, with $c(v) \neq i$.



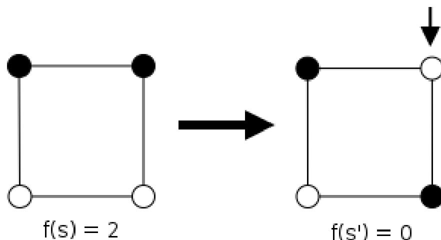
Defining Neighborhood: GCP

Swap move

It consists of swapping the color of a single conflicting vertex v with the color i ($c(v) \neq i$) of a non-conflicting adjacent vertex w.

Swap move

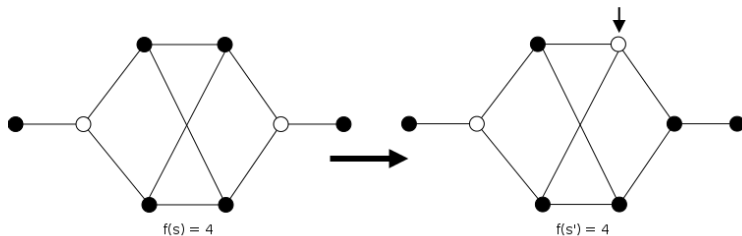
It consists of swapping the color of a single conflicting vertex v with the color i ($c(v) \neq i$) of a non-conflicting adjacent vertex w.



Defining Neighborhood: GCP

Swap move

It consists of swapping the color of a single conflicting vertex v with the color i ($c(v) \neq i$) of a non-conflicting adjacent vertex w.



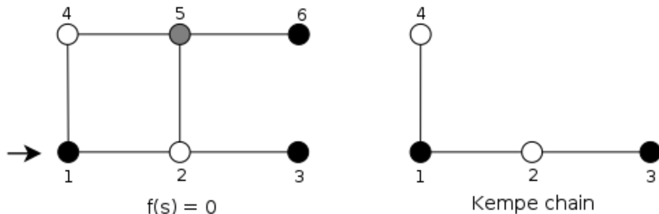
Defining Neighborhood: GCP

Kempe chain move

Let a and b be two colors. The (a,b)-Kempe chain of G is the set of nodes that form a connected component in the subgraph of G induced by the nodes that are colored either a or b.

Kempe chain move

Let a and b be two colors. The (a,b)-Kempe chain of G is the set of nodes that form a connected component in the subgraph of G induced by the nodes that are colored either a or b.



The chain for the case that we want to color node 1 white.

Defining Neighborhood: GCP

Kempe chain move

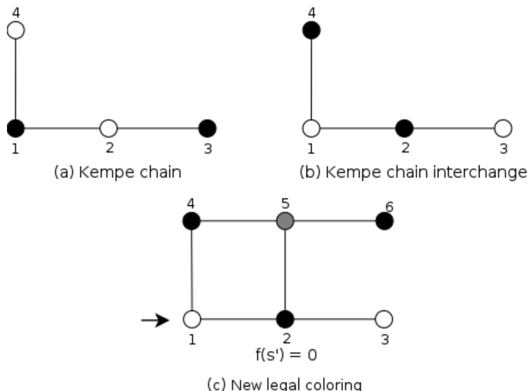
Let a and b be two colors. The (a,b)-Kempe chain of G is the set of nodes that form a connected component in the subgraph of G induced by the nodes that are colored either a or b.

➥ To construct a new coloring we perform a Kempe chain interchange. This means that we swap the colors in the chain.

Defining Neighborhood: GCP

Kempe chain move

Let a and b be two colors. The (a,b)-Kempe chain of G is the set of nodes that form a connected component in the subgraph of G induced by the nodes that are colored either a or b.



(a) Kempe chain

(b) Kempe chain interchange

(c) New legal coloring

Partial k-coloring and Neighborhoods

Partial k-coloring neighborhood:

- A solution is a partition of vertices of G into k disjoint color classes $\{V_1, \ldots, V_k\}$. We use a specific class (i.e., V_{k+1}) to represent the set of uncolored vertices.
- A neighbor solution is obtained by moving an uncolored vertex u from V_{k+1} to a pre-existing color class V_h, and by moving to V_{k+1} all vertices in V_h that are adjacent to u.
- The complete legal k-coloring is obtained by emptying V_{k+1}.

Simple Mechanisms for Escaping from Local Optima

Issue: how to avoid getting trapped in bad local optima?

- **Restart**: re-initialize search whenever a local optimum is encountered.
  (Often rather ineffective due to cost of initialization.)
- **Non-improving steps**: in local optima, allow selection of candidate solutions with equal or worse evaluation function value, e.g., using minimally worsening steps.
  (Can lead to long walks in large plateaus, i.e., regions of search positions with identical evaluation function.)
- Use **more complex neighborhoods**

➥ **Note**: Neither of these mechanisms is guaranteed to always escape effectively from local optima.

Diversification vs Intensification

- Goal-directed and randomized components of LS strategy need to be balanced carefully.
- **Intensification**: aims to greedily increase solution quality e.g., by exploiting the evaluation function.
  ➥ Iterative Improvement (II): intensification strategy
- **diversification**: aim to prevent search stagnation by preventing search process from getting trapped in confined regions.
  ➥ Uninformed Random Walk (URW): diversification strategy

Balanced combination of intensification and diversification mechanisms forms the basis for advanced LS methods.

Trajectory Based Metaheuristics

- Stochastic Local Search
- Simulated Annealing
- Iterated Local Search
- Tabu Search
- Variable Neighborhood Search
- Guided Local Search

Randomized Iterative Improvement (RII) aka, Stochastic Hill Climbing

Principe général : À chaque itération, effectuer un move alétoire avec une certaine probabilité p au lieu de toujours choisir une solution de meileure qualité.

Algorithme 3: Randomized Iterative Improvement (RII):

determine initial candidate solution s;
while *termination condition is not satisfied* **do**
  r ← Random(0, 1) ;
  if (r < p) **then**
    s' ← a neighbor of s chosen uniformly at random ;
  else
    choose a neighbor s' of s such that $f(s') < f(s)$ or ;
    if no such s' exists, choose s' such that $f(s')$ is minimal
  end
  s ← s'
end

Randomized Iterative Improvement (RII)

Observations centrales:

- Plus besoin de terminer lorsqu'on tombe sur un optimum local
  - Fixer une borne sur le **temps CPU**, **nombre d'itérations**
  - Fixer une borne sur le temps (ou nombre d'itérations) **après la dernière amélioration du best**

- Un choix probabiliste permet d'avoir une séquence arbitrairement longue d'étapes de '**marche aléatoire**'
  - Lorsque exécuté suffisament longtemps, RII trouve une solution (optimale) avec une probabilité arbitraiement elevée

➥ On parle de convergence

Randomized Iterative Best Improvement for SAT

Une variante de cet algorithme a longtemps été l'état de l'art pour SAT (GWSAT, Selman et al., 1994)

Algorithme 4: GSAT with Random Walk (GWSAT)

Choisir une instanciation A aléatoire uniforme des variables de F ;
Step $\leftarrow 0$;
tant que *NON*(A *satisfait* F)&&(*Steps* $<$ *maxSteps*) **faire**
    Avec probabilité p **faire**
        Choisir de façon aléatoire uniforme une variable x de F ;
    sinon
        Choisir de façon aléatoire uniforme une variable x de F tel que changer la valeur de x dans A diminue le nombre de clauses non vérifiées de F de facon maximale ;
    finsi
    Changer la valeur de x **dans** A ;
    Step $\leftarrow$ *step* $+ 1$;
fin
si (A satisfait F) **alors** retrouner A ;

Probabilistic Iterative Improvement (PII)

Principe général :

- Accepter des 'move' vers des solutions moins bonnes, avec une certaine probabilité qui dépend de la déterioration de la fonction d'évaluation

  ➥ Une plus grande déterioration → Une probabilité plus petite

- Une fonction $p(f, s)$: détermine la distribution de probabilité sur les voisins de s en fonction de la fonction d'évaluation f
- À chaque étape de recherche, sélectionner simplement un voisin de la solution actuelle selon p
- La nature de la recherche dépend fortement du choix de la distribution de probabilité $p(f, s)$

Metropolis PII pour TSP

- **Espace de recherche** : les cycles hamiltoniens dans un graphe donnée
- **Représentation** : permutations des villes
- **Fonction de voisinage** $N_{\mathcal{I}}(s)$: 2-edge-exchange
- **Initialisation** : un CH choisit aléatoirement (une permutation)
- **Move** :
  1. Choisir aléatoirement une solution s' dans $N_{\mathcal{I}}(s)$
  2. Accepter s' avec une probabilité $p(T, s, s')$ où T est un paramètre (Metropolis Condition):

$$p(T, s, s') = \begin{cases} 1 & \text{if } f(s') \leq f(s) \\ exp^{\frac{-(f(s')-f(s))}{T}} & \text{otherwise} \end{cases}$$

- **Terminaison** : temps CPU

Simulated Annealing

- Mimer le processus physique de refroidissement de matériaux
  - En chauffant de la matière, puis en la refroidissant très doucement, on peut obtenir des structures cristallines extrêment résistantes

- **Origine** :
  - Metropolis (1953)
  - Optimization by Simulated Annealing (KirkPatrick, 1983)

- **Principe général** : Varier la température (i.e. la probabilité d'accepter une solution de moins bonne qualité), dans une PII suivant une fonction de refroidissement (Annealing schedule /Cooling)

Algorithme 5: Simulated Annealing

determine initial candidate solution s ;
set initial temperature T according to annealing schedule ;
while *termination condition is not satisfied* **do**
    while *maintain same temperature* T *according to annealing schedule* **do**
        probabilistically choose a neighbor s' of s using proposal mechanism ;
        if s' *satisfies probabilistic acceptance criterion (depending on T)* **then**
            s ← s';
        end
    end
    update T according to annealing schedule ;
end

- proposal mechanism: often uniform random choice from $N_{\mathcal{I}}(s)$
- acceptance criterion: often Metropolis condition
- annealing Schedule: valeur de T en fonction de l'avancement de la recherche

Exemple d'instanciation SA

Algorithme 6: Simulated Annealing

s $\leftarrow$ une solution initiale ;
T $\leftarrow$ T_{max} une température initiale ;
répéter
    répéter
        s' $\leftarrow$ un voisin de s choisit aléatoirement;
        $\delta \leftarrow f(\text{s}) - f(\text{s'})$;
        si $(\delta \geq 0)$ **alors** s $\leftarrow$ s';
        sinon s $\leftarrow$ accepter s' avec probabilité $exp^{(\delta/T)}$;
    jusqu'à *(Condition d'équilibre)*;
    T $\leftarrow$ $g(T)$;
jusqu'à *(Condition d'arrêt)*;

Retourner la meilleure solution rencontrée

Move Acceptance

- **Metropolis condition**
  - $exp^{(\delta/T)}$
- **Rôle de la température**
  - $T <<<$: recherche locale (plutôt à la fin)
    ➠ Faible probabilité d'accepter un 'mauvais' move
  - $T >>>$: marche probabiliste (plutôt au début)
    ➠ Probabilité plus forte d'accepter un 'mauvais' move
- **Rôle de l'évaluation**
  - À T fixe, plus la différence de qualité des solutions est petite, plus la probabilité d'accepter est grande

État d'équilibre

- Nombre d'évaluations à effectuer à chaque température
  - En général fonction de la taille du voisinage
- Approche statique (nombre d'itération/évaluation) fixé au début
- Approche adaptative
  ➠ Petit (resp. grand) nombre d'évaluations à température chaude (resp. froide)

Cooling

- Compromis entre la qualité de la solution et le temps de calcul

- Plusieurs stratégies
  - Linéaire : $T_{i+1} = T_0(1 - i/\beta)$
  - Géométrique : $T_{i+1} = \alpha.T_i$
  - Logarithmique: $T_i = T_0/\log(i)$

Initialisation et terminaison

- Une température suffisament chaude mais pas trop !
  - Typiquement avec une solution initiale gloutonne
- Condition d'arrêt
  - Température minimale atteinte
  - Nombre d'itérations (temps de calcul) sans amélioration

Tabu Serach

Idée principe :

- Utiliser une mémoire au cours de la recherche pour échapper des optima locaux
- Associer une liste tabou respectivement à certaines solutions ou à certaines composantes
- **Ne pas autoriser de visiter des solutions (ayant des composantes) dans la liste tabu**

Algorithme 7: Tabu search

determine initial candidate solution s ;
while *termination condition is not satisfied* **do**
  determine set N' of non-tabu neighbors of s ;
  choose a best candidate solution s' in N' ;
  update tabu attributes based on s' ;
  s $\leftarrow$ s';
end

- À chaque étape de la recherche
  - la solution, ou les composants ajoutés/supprimés, sont déclarés tabous pour un nombre fix/'e d'étapes de recherche $\rightarrow$ Tabu Tenure
- **Critère d'aspiration**
  - Définit des exceptions qui peuvent amener la recherche à ne pas respecter les movements tabu

TS: The minimum weighted k-cardinality tree problem

The minimum weighted k-cardinality tree problem (k-CARD for short) Given an undirected weighted graph $G = (V, E)$, The problem consists in finding a subtree $T = (V_T, E_T)$ of G with exactly k edges whose sum of weights is minimum.

For each $e \in E$, a weight $w_e \in R$ is given. With $w(T)$ we denote weight of $T = (V_T, E_T)$, such that

$$w(T) = \sum_{e \in E_T} w_e, \forall E_T \subseteq E.$$

More formally, the k-CARD problem is

$$\min\{w(T) | T = (V_T, E_T) \text{ is a tree}, |E_T| = k\}$$

➥ This problem is known to be strongly NP-hard.

TS: The minimum weighted k-cardinality tree problem

Solution space
A solution space $\mathcal{S}$ for the k-CARD problem can be represented as a set of all subtrees of $G = (V,E)$ with exactly k edges (or $k+1$ nodes).

Neighborhood function : edge-swapping
replace a selected edge in the tree by another selected edge outside the tree, subject to requiring that the resulting subgraph is also a tree.

Two types of such edge swaps:

- one that maintains the current nodes of the tree unchanged (static)
- one that results in replacing a node of the tree by a new node (dynamic)

TS: The minimum weighted k-cardinality tree problem

Tabu classification: Associate tabu status for an **added/dropped edge**

- For an **added edge**, tabu-active means that this edge is not allowed to be dropped from the current tree for the number of iterations that defines its tabu tenure.
- For a **dropped edge**, tabu-active means that **this edge is not allowed to be included in the current solution** during its tabu tenure.
- The tabu tenure for added edges is bounded by k, since if no added edge is allowed to be dropped for k iterations, then within k steps all available moves will be classified tabu.

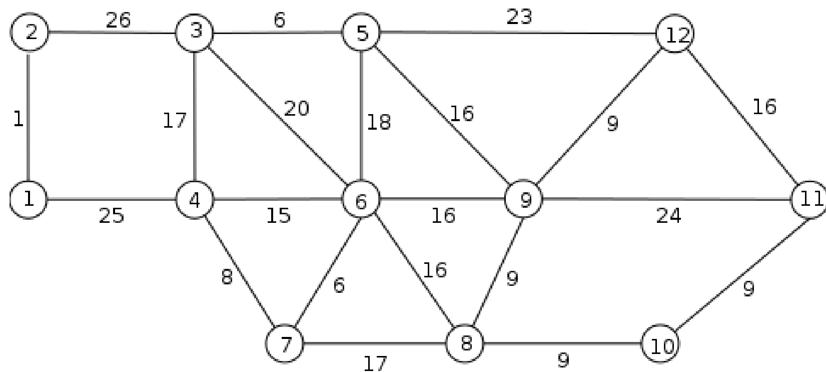
➟ Keep a recently dropped edge tabu-active for a large period time than a recently added edge.

TS: The minimum weighted k-cardinality tree problem

Initialization: a greedy heuristic

- The greedy construction starts by choosing the edge (i, j) with the smallest weight in the graph, where i and j are the indexes of the nodes that are the endpoints of the edge.
- The remaining $(k - 1)$ edges are chosen successively to minimize the increase in total weight at each step, where the edges considered meet exactly one node from those that are endpoints of edges previously chosen.

Exemple of Four-cardinality problem



Initialization: compute an initial solution

➥ The objective function value is equal to 40.

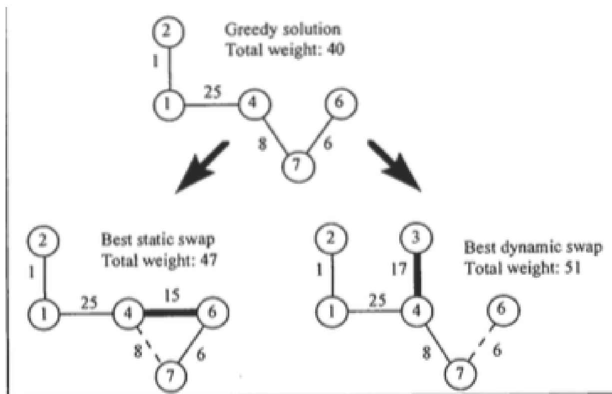
Table 2.2 Greedy construction.

Step	Candidates	Selection	Total Weight
1	(1,2)	(1,2)	1
2	(1,4), (2,3)	(1,4)	26
3	(2,3), (3,4), (4,6), (4,7)	(4,7)	34
4	(2,3), (3,4), (4,6), (6,7), (7,8)	(6,7)	40

TS: The minimum weighted k-cardinality tree problem

Iteration 1: Swap move types

➥ The objective function value is equal to 47



TS: The minimum weighted k-cardinality tree problem

Iteration 1: Tabu Classification

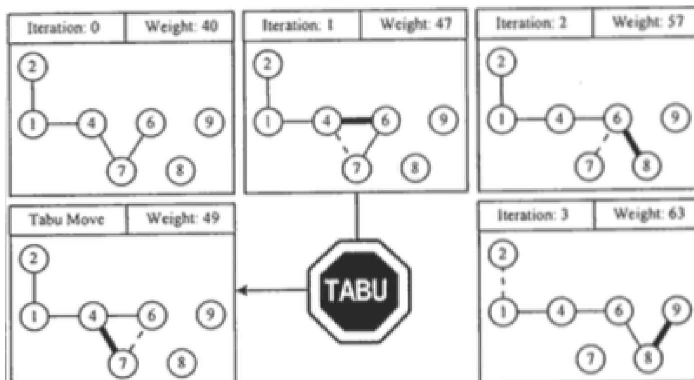
➥ Values of 1 and 2 for the currently tabu-active edges indicate the number of iterations that these edges will remain tabu-active

Iteration	Tabu-active net tenure		Add	Drop	Weight
	1	2			
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	63

TS: The minimum weighted k-cardinality tree problem

Iteration 2: The tabu-active classification of edge $(4, 7)$ has modified the original neighborhood of the solution at iteration 2, and has forced the search to choose a move with an inferior objective function value

➥ The objective function value is now equal to 57.

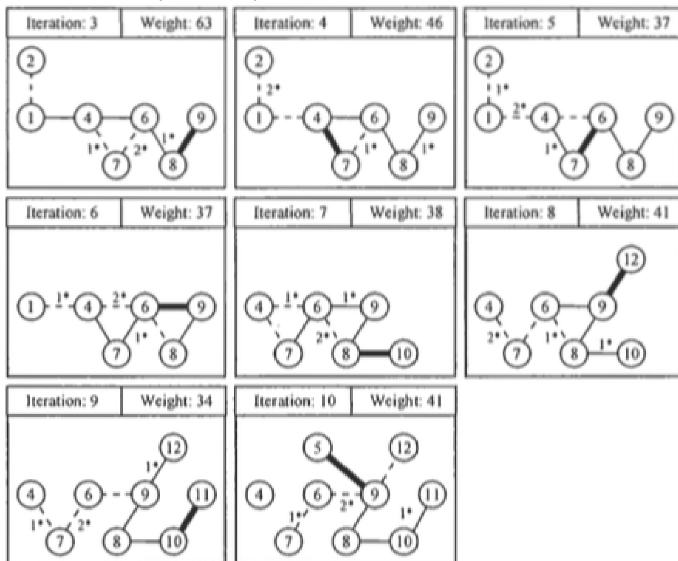


Iterations of TS procedure: Tabu Classification

Iteration	Tabu-active net tenure		Add	Drop	Move Value	Weight
	1	2				
3	(6,8), (4,7)	(6,7)	(8,9)	(1,2)	6	63
4	(6,7), (8,9)	(1,2)	(4,7)	(1,4)	-17	46
5	(1,2), (4,7)	(1,4)	(6,7)	(4,6)	-9	37*
6	(1,4), (6,7)	(4,6)	(6,9)	(6,8)	0	37
7	(4,6), (6,9)	(6,8)	(8,10)	(4,7)	1	38
8	(6,8), (8,10)	(4,7)	(9,12)	(6,7)	3	41
9	(4,7), (9,12)	(6,7)	(10,11)	(6,9)	-7	34*
10	(6,7), (10,11)	(6,9)	(5,9)	(9,12)	7	41

TS: The minimum weighted k-cardinality tree problem

Graphical representation of TS iterations



Iterated Local Search (ILS)

Idée principe :

- Recherche locale itérative pour atteindre rapidement un optimum local (Intensification)
- Perturbation pour échapper aux optimas locaux (Diversification)
- Critère d'acceptation pour contrôler la balance entre diversification et intensification

ILS peut être vue comme une marche dans l'espace des Optima locaux

- La recherche locale définit en quelque sorte un voisinage (on parle de bassins d'attraction)

Les différentes composantes du ILS sont complémentaires

Algorithme 8: ILS

determine initial candidate solution s ;
perform subsidiary local search on s **while** *termination condition is not satisfied* **do**

    s' ← `Perturbation`(s) ;
    s'' ← `LocalSearch`(s') ;
    s ← `Acceptance`(s, s'') ;

end

ILS – Perturbation mechanism

- Needs to be chosen such that its effect *cannot* be easily undone by subsequent local search phase (Often achieved by search steps larger neighborhood)
  Example: local search = 3-opt, perturbation = 4-exchange steps in ILS for TSP

- A perturbation phase may consist of one or more perturbation steps

- Weak perturbation ➠ short subsequent local search phase
  but: risk of revisiting current local minimum

- Strong perturbation ➠ more effective escape from local minima
  but: may have similar drawbacks as random restart

- Advanced ILS algorithms may change nature and/or strength of perturbation adaptively during search

ILS – Acceptence criteria

- Always accept the best of the two candidate solutions

  ➠ ILS performs Iterative Improvement in the space of local optima reached by subsidiary local search

- Always accept the most recent of the two candidate solutions

  ➠ ILS performs random walk in the space of local optima reached by subsidiary local search

- Intermediate behavior: select between the two candidate solutions based on the Metropolis criterion

- Advanced acceptance criteria take into account search history

Variable Neighborhood Search (VNS)

(Hansen & Mladenovic, 1997)

Variable Neighborhood Search is an SLS method which systematically exploits the idea of **neighborhood change**, both in descent to local minima and in escape from the valleys which contain them.

Central observations

- a local minimum w.r.t. one neighborhood structure is not necessarily locally minimal w.r.t. another neighborhood structure.
- a global optimum is locally optimal w.r.t. **all** neighborhood structures.

Variable Neighborhood Search (VNS)

- **Key idea:** systematic change of the neighborhood during the search
- Several adaptations of this central principle
  (a) Variable Neighborhood Descent (VND)
  (b) Reduced Variable Neighborhood Search (RVNS)
  (c) Basic Variable Neighborhood Search (BVNS)
  (d) General Variable Neighborhood Search (GVNS)
  (e) Variable Neighborhood Decomposition Search (VNDS)
  (f) Skewed Variable Neighborhood Search (SVNS)
- Notations
  - N_k, $k = 1, 2, \ldots, k_{max}$ is a set of neighborhood structures
  - $N_k(s)$, is the set of solutions in the k-th neighborhood of s

Algorithme 9: NeighbourhoodChange

```
function NeighbourhoodChange (s, s', k);
begin
1   if f(s') < f(s) then
2       s ← s' ; k ← 1 /*Make a move*/ ;
    end
3   else k ← k + 1 /*Next neighbourhood*/ ;
4   return;
end
```

The change of the neighborhoods can be performed in three different ways:

(i) deterministic;

(ii) stochastic;

(iii) both deterministic and stochastic.

Variable Neighborhood Descent (VND)

Algorithme 10: Variable Neighborhood Descent

```
Function VND (s, k_max);
begin
1   repeat
2       k ← 1;
3       repeat
4           s' ← argmin_{y∈N_k(s)} f(y) /*Find the best neighbor in Nk(s) */ ;
5           NeighbourhoodChange (s,s',k) ;
        until k = k_max;
    until Termination Condition;
end
```

➥ The change of neighbourhoods is performed in a deterministic way.

Variable Neighborhood Descent (VND)

- Final solution is locally optimal w.r.t. all neighborhoods.
  ➥ the chances to reach a global optimum are larger when using VND than with a single neighborhood structure.

- First improvement may be applied instead of best improvement.

- Typically, order neighborhoods from smallest to largest.

- The stopping condition:
  - the maximum cpu time allowed t_{max},
  - the maximum number of iterations between two improvements.

Reduced Variable Neighborhood Search (RVNS)

Method useful in very large instances, for which local search is costly

- Only explores randomly different neighborhoods
  A solution is selected randomly (instead of a descent) from $N_k(s)$.
- The best value for the parameter k_{max} is often 2
- The maximum number of iterations between two improvements is usually used as a stopping condition
- Can be faster than standard local search algorithms for reaching good quality solutions

Algorithme 11: Steps of Reduced VNS

```
    Function RVNS (s, k_max, t_max);
    begin
1       repeat
2           k ← 1;
3           repeat
4               s' ← Shake (s, N_k);
5               NeighbourhoodChange (s, s', k) ;
            until k = k_max;
6           t ← CpuTime ()
        until t > t_max;
    end
```

Mixed methods

Previous methods can be improved by combining both deterministic and stochastic neighborhood change

- Select a initial solution
- Find a direction of descent from s in $N(s)$
- Move to the minimum within $N(s)$

Basic Variable Neighborhood Search (BVNS)

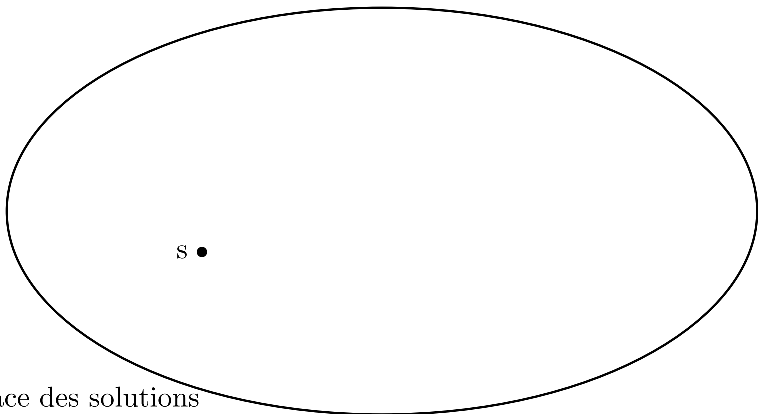
Algorithme 12: Steps of the Basic VNS

```
    Function RVNS (s, k_max, t_max);
    begin
1       repeat
2           k ← 1;
3           repeat
4               s' ← Shake (s,N_k) /* Shaking */ ;
5               s'' ← FirstImprovement (s') /* Local search */ ;
6               NeighbourhoodChange (s,s'',k) /*Change neighbourhood*/ ;
            until k = k_max;
7           t ← CpuTime ()
        until t > t_max;
    end
```

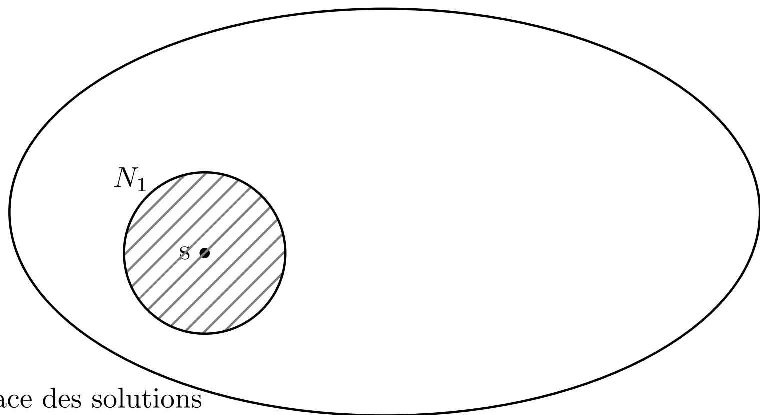
Two important observations

- Solution s' is generated at random in Step 4 in order to avoid cycling, which might occur if a deterministic rule were applied.
- In Step 5, the first improvement local search (Algorithm 2) can be replaced with best improvement (Algorithm 1).

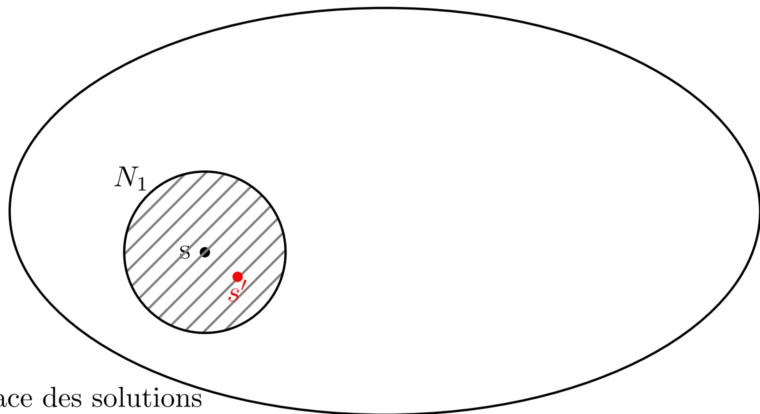
Basic Variable Neighborhood Search (BVNS)



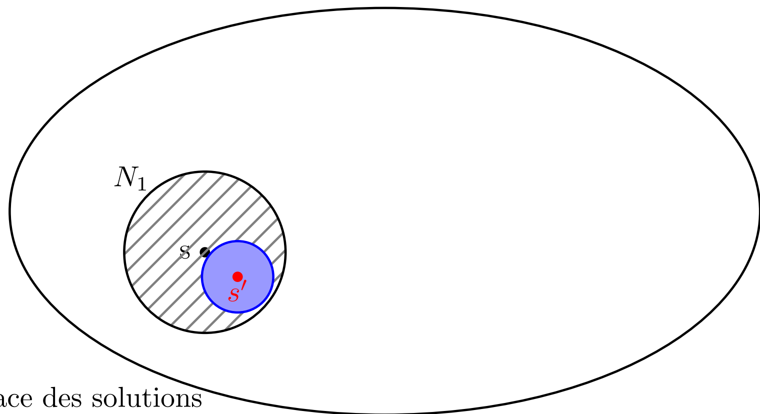
Basic Variable Neighborhood Search (BVNS)



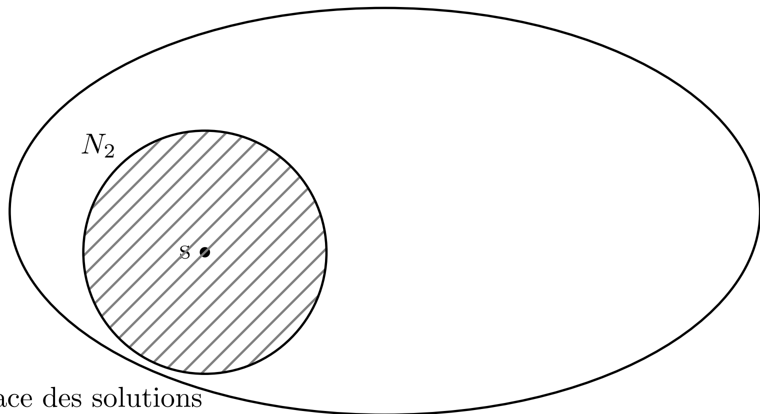
Basic Variable Neighborhood Search (BVNS)



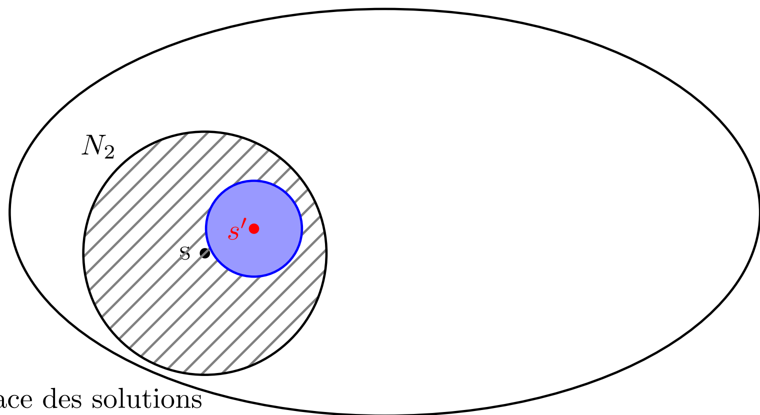
Basic Variable Neighborhood Search (BVNS)



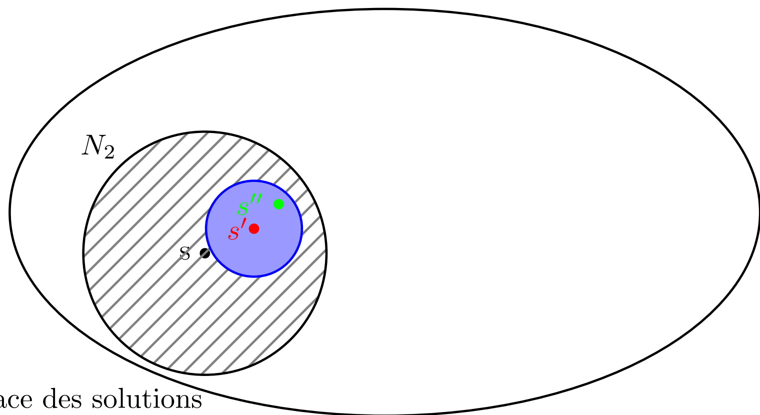
Basic Variable Neighborhood Search (BVNS)



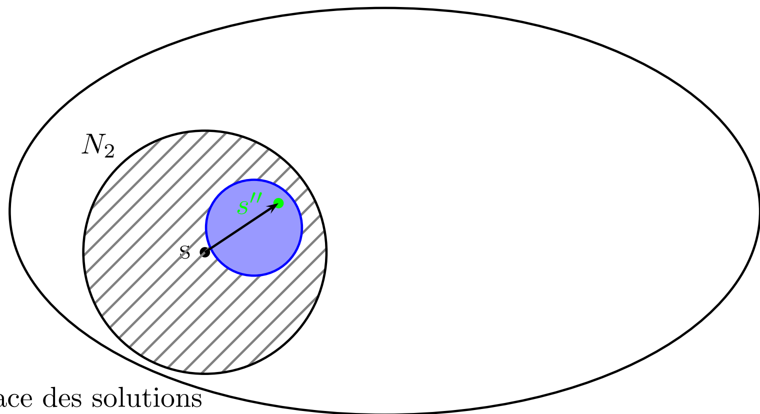
Basic Variable Neighborhood Search (BVNS)



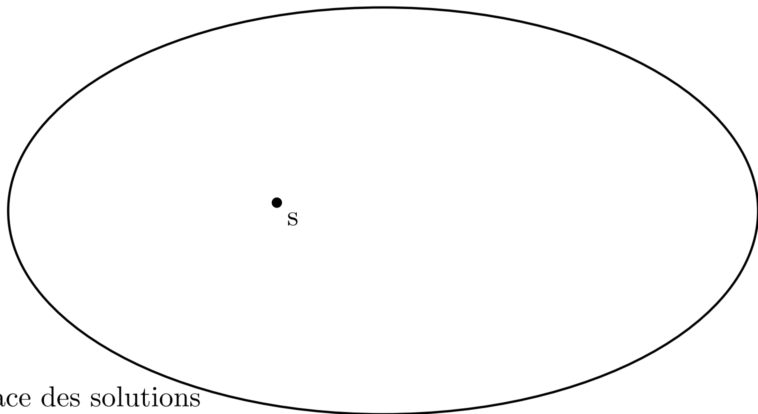
Basic Variable Neighborhood Search (BVNS)



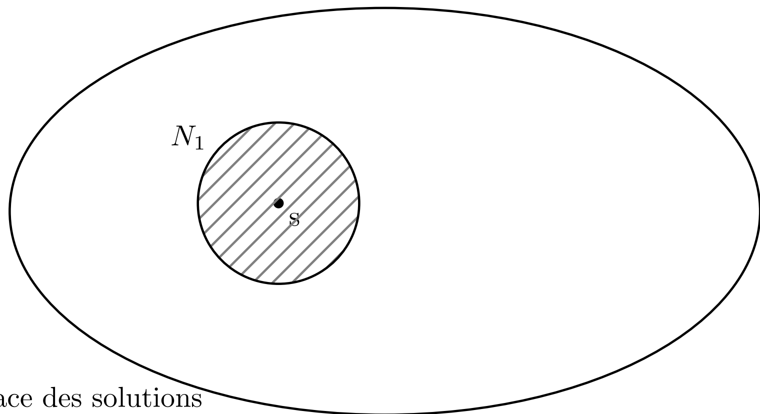
Basic Variable Neighborhood Search (BVNS)



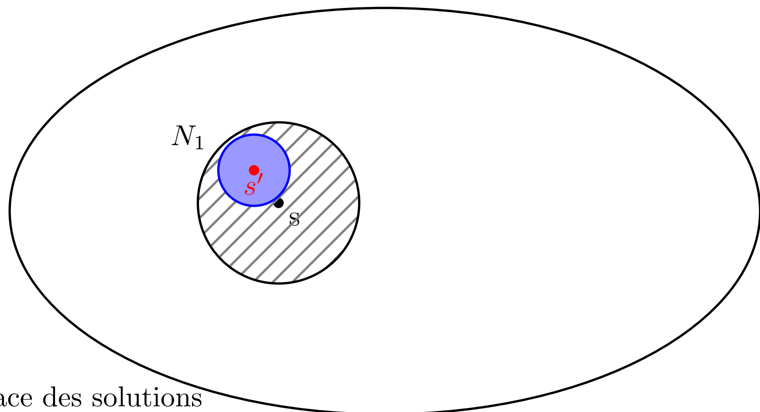
Basic Variable Neighborhood Search (BVNS)



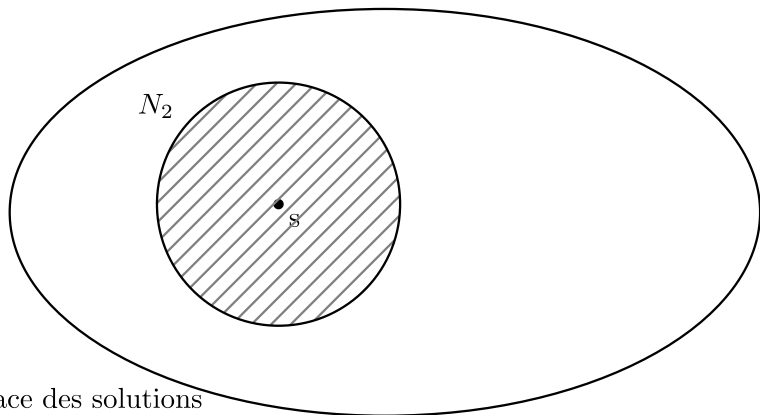
Basic Variable Neighborhood Search (BVNS)



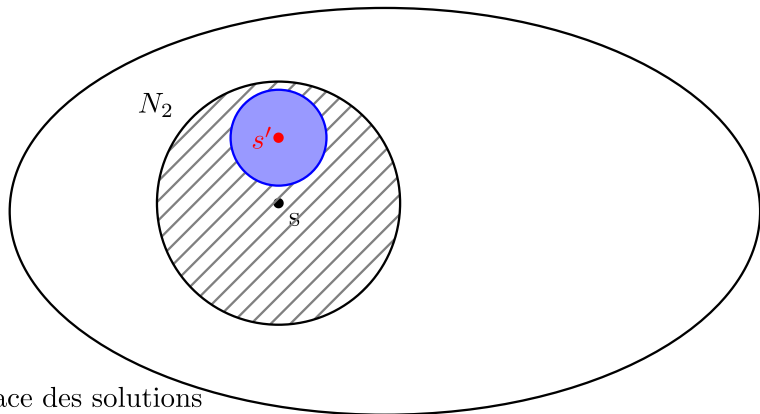
Basic Variable Neighborhood Search (BVNS)



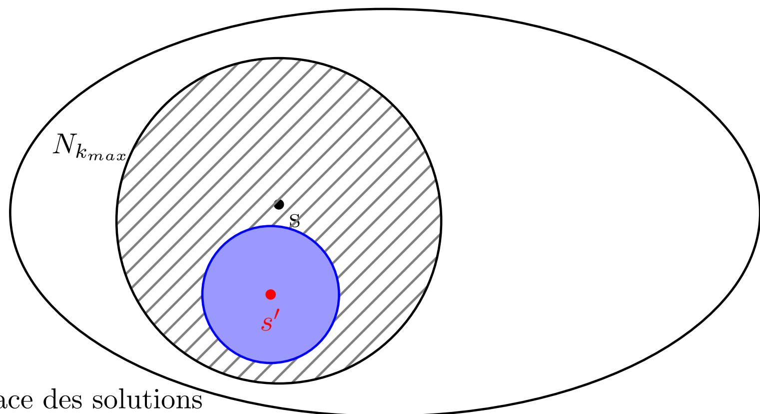
Basic Variable Neighborhood Search (BVNS)



Basic Variable Neighborhood Search (BVNS)



Basic Variable Neighborhood Search (BVNS)



BVNS: The minimum weighted k-cardinality tree problem

The minimum weighted k-cardinality tree problem (k-CARD for short)
Given an undirected weighted graph $G = (V,E)$, The problem consists in finding a subtree $T = (V_T,E_T)$ of G with exactly k edges whose sum of weights is minimum.

For each $e \in E$, a weight $w_e \in R$ is given. With $w(T)$ we denote weight of $T = (V_T,E_T)$, such that

$$w(T) = \sum_{e \in E_T} w_e, \forall E_T \subseteq E.$$

More formally, the k-CARD problem is

$$\min\{w(T) | T = (V_T, E_T) \text{ is a tree}, |E_T| = k\}$$

➥ This problem is known to be strongly NP-hard.

BVNS: The minimum weighted k-cardinality tree problem

Solution space

A solution space $\mathcal{S}$ for the k-CARD problem can be represented as a set of all subtrees of $G = (V, E)$ with exactly k edges (or $k + 1$ nodes).

Neighborhood structure

Let $\rho(T_1, T_2)$ define the distance between any two solutions (trees with cardinality k) T_1 and T_2 as a cardinality of difference between their edge sets, i.e.,

$$\rho(T_1, T_2) = |E_{T_1} \setminus E_{T_2}| = |E_{T_2} \setminus E_{T_1}|.$$

The neighborhood $N_p(T)$ consists of all solutions (subtrees) with distance p from T:

$$N_p(T) = \{T' |\ \rho(T, T') = p\}$$

Local Search

The local search is performed in the N_1 neighborhood of the current solution T.

In other words, given a tree T, we generate a new $T' \in N_1(T)$ by selecting one edge from the set of edges not in T and removing one edge in the tree:

- Given a tree T, a new tree $T' \in N_1(T)$ is generated by selecting pairs of vertices (v_{out}, v_{in}), $v_{out} \in V_T$, $v_{in} \in V \setminus V_T$, and interchange them if T' is connected.
- If $w(T') < w(T)$, a move is made ($T \leftarrow T'$); otherwise T is local minimum, and the local search is terminated.

➥ The way for selecting T' can be perfomed either by first or best improvement strategies.

Shaking step

To generate a random solution T' from neighborhood $N_p(T)$, the following steps are repeated p times:

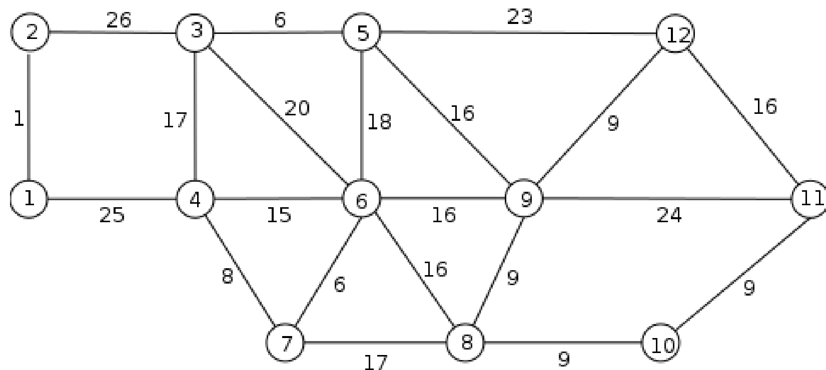
(i) Find the set of nonarticulation[1] points of the graph induced by the current set of vertices V_T, and select one among them at random that belongs to the root tree T ($v_{out} \in V_T$).

(ii) Find the set of vertices not in the current V_T that are connected with $V_T \setminus \{v_{out}\}$, and select one at random that does not belong to the root tree T ($v_{in} \notin V_T$).

(iii) A new set of vertices is obtained by the interchange: $V_T \leftarrow V_T \setminus \{v_{out}\} \cup \{v_{in}\}$.

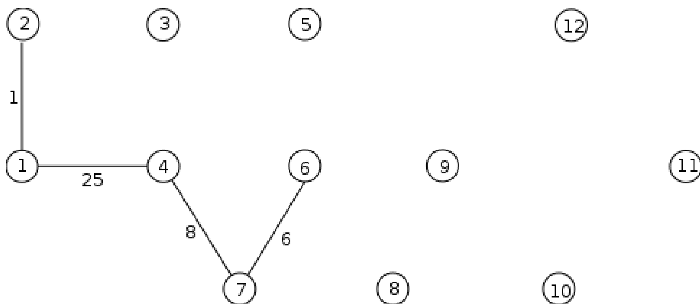
[1] A vertex v is an articulation point of a graph if its removal disconnects the graph.

Exemple of Four-cardinality problem



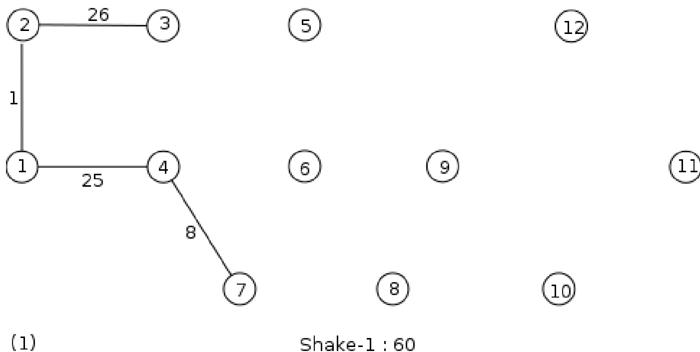
Step 1: compute an initial solution

➥ The objective function value is equal to 40.



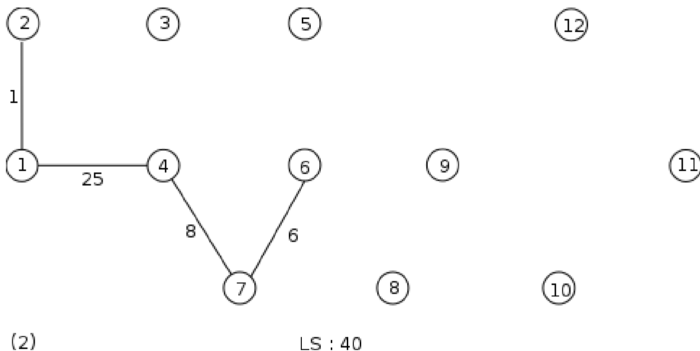
Step 2: shaking step with p=1

➥ The objective function value is equal to 60.



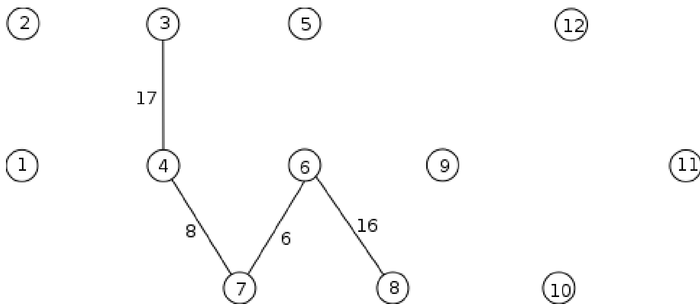
Step 3: local search step

➥ We return to the same solution.



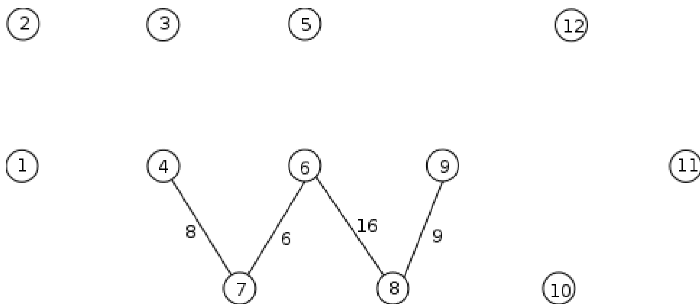
Step 4: shaking step with p=2, i.e., we take out 2 edges and add another 2 edges at random

➥ The objective function value is now equal to 47.



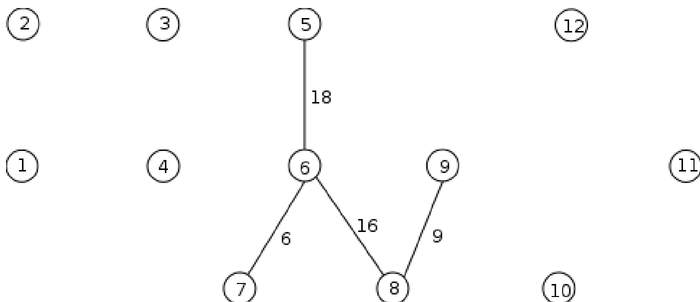
Step 5: local search step

➥ An improved solution is obtained with a value of 39.



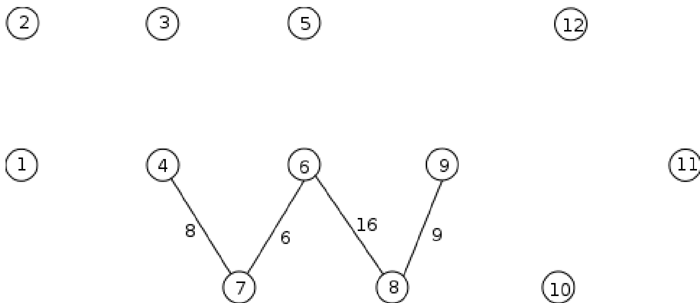
Step 6: shaking step with p=1

➥ A solution of cost 49 is obtained.



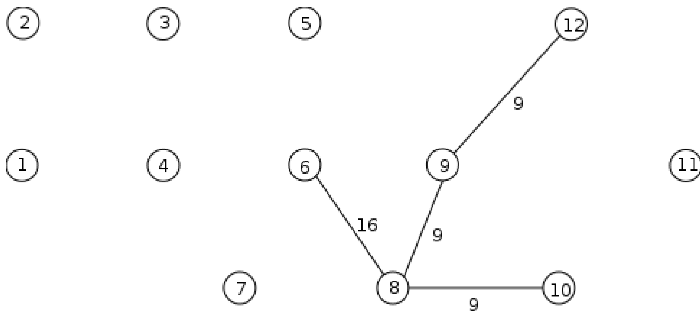
Step 7: local search step

➥ We return to the same solution.



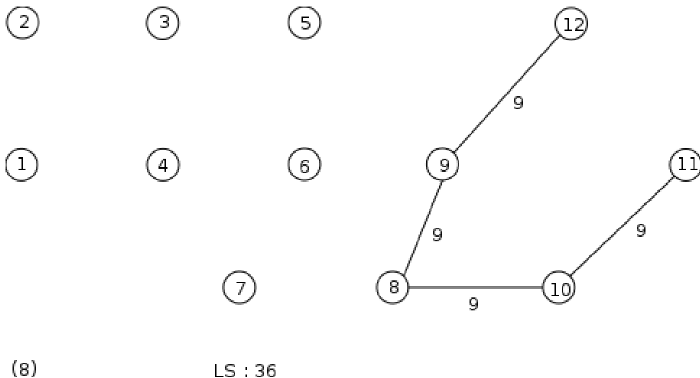
Step 8: shaking step with p=2

➥ A solution is obtained with a value of 43.



Step 9: local search step

➥ we find the optimal solution with an objective function value equal to 36.



Skewed Variable Neighborhood Search (SVNS)

- Set $s \leftarrow s''$ even if s'' it is worse
  - Skewed VNS: accept if $f(s'') - \alpha \times \rho(s, s'') < f(s)$
    $\rho(s, s'')$ measures the distance between solutions

- The parameter α must be chosen in order to accept the exploration of valleys far away from s when $f\ (s'')$ is larger than $f(s)$.

- To avoid frequent moves from s to a close solution, one may take a large value for α when $\rho(s, s'')$ is small.

➥ SVNS provides a useful mechanism to diversify the search when one region of the solution space with good solutions is far from another.

Algorithme 13: Steps of neighbourhood change for the skewed VNS

```
function NeighbourhoodChangeS (s, s'', k, α);
begin
1   if f(s'') − α × ρ(s, s'') < f(s) then
2       s ← s'' ; k ← 1;
    end
3   else  k ← k + 1;
4   return;
end
```

Algorithme 14: Steps of the Skewed VNS

```
Function SVNS (s, k_max, t_max, α);
begin
1     repeat
2         k ← 1, s_best ← s ;
3         repeat
4             s' ← Shake (s,N_k) ;
5             s'' ← FirstImprovement (s') ;
6             KeepBest (s_best,s) ;
7             NeighbourhoodChangeS (s,s'',k) ;
          until k = k_max;
8         s ← s_best ;
9         t ← CpuTime ()
      until t > t_max;
end
```

Variable Neighborhood Decomposition Search (VNDS)

Central idea:

- generate subproblems by keeping all but k solution components fixed
- apply a local search only to the "free" components

Variable Neighborhood DEcomposition Search (VNDS)

Algorithme 15: Steps of the VNDS

```
Function VNDS (s, k_max, t_max, α);
begin
1    repeat
2        k ← 2;
3        repeat
4            s' ← Shake (s, N_k) ;
5            y ← s' \ s ;
6            y' ← VNS (y, k, t_max) ;
7            s'' ← (s' \ y) ∪ y' ;
8            s''' ← FirstImprovement (s'') ;
9            NeighbourhoodChange (s, s''', k) ;
         until k = k_max;
10       t ← CpuTime ()
     until t > t_max;
end
```

Exploration des voisinages étendus par de la PPC

Pourquoi les voisinages étendus?

Voisinages étendus : Avantages

- A **larger neighborhood** means:
  - More solutions are considered
  - Better chance of avoiding local minima

Voisinages étendus : Inconvénients

- A **larger neighborhood** also means:
  - More solutions need to be evaluated
  - The complexity of evaluating all solutions makes having neighborhoods too large unattractive

- Unless we do not evaluate all the solutions !
  - This is were Constraint Programming is useful

Constraint Programming vs. Local Search

- Local search methods
  - Very fast and efficient
  - Model and search strategies closely linked
  - Complex constraints hard to model
- Constraint Programming
  - The traditional Depth-First Search strategy is too slow
  - Model and Search are completely separated
  - Complex constraints fairly easy to model
- **The ultimate goal would be to get all the advantages without the inconveniences**

Hybridations sur la base d'un algorithme de RL

Objectif: Utiliser des mécanismes complets pour mieux exploiter la notion de voisinage et de choix dans le voisinage.

- **Exploration des voisinages par une recherche complète**:
  - Relaxer (désinstancier) une grande partie d'une instanciation complète courante, puis essayer de la reconstruire si possible de meilleure qualité.
  - Large Neighborhood Search (LNS, Show98) utilise un LDS pour explorer le voisinage étendu.
  - PGLNS Propagation Guided LNS (Perron et al., 2004), utilise une heuristique de choix de voisinage basée sur la notion d'impact [1].

→ C'est de loin les hybridations les plus fructueuses.

[1] L'impact d'une variable correspond au nombre de valeurs filtrées de son domaine.

VNS/LDS+CP

(Loudni & Boizumault, EJOR 2008)

Un *schéma d'hybridation imbriquée sur la base d'une RL* :

- une recherche locale dans un voisinage de taille variable (VNS), à base de relaxation/reconstruction d'un sous-ensemble des variables du problème;
- une reconstruction à base de LDS combinée avec une propagation de contraintes à base de calcul de minorant, pour évaluer le coût et la légalité des *mouvements* effectués.

VNS/LDS+CP

(Loudni & Boizumault, EJOR 2008)

Pour k une dimension de voisinage, et s une solution courante :

VNS/LDS+CP

(Loudni & Boizumault, EJOR 2008)

Pour k une dimension de voisinage, et s une solution courante :

- N_k correspond à l'ensemble des combinaisons de k variables parmi X,

Pour k une dimension de voisinage, et s une solution courante :

- N_k correspond à l'ensemble des combinaisons de k variables parmi X,
- on sélectionne dans N_k un sous-ensemble de k variables, noté X_{un},

VNS/LDS+CP

(Loudni & Boizumault, EJOR 2008)

Pour k une dimension de voisinage, et s une solution courante :

- N_k correspond à l'ensemble des combinaisons de k variables parmi X,
- on sélectionne dans N_k un sous-ensemble de k variables, noté X_{un},
- une affectation partielle A_k est définie en désaffectant les k variables : $A_k = s \setminus \{(x_i = v_i)\ t.q.\ x_i \in X_{un}\}$,

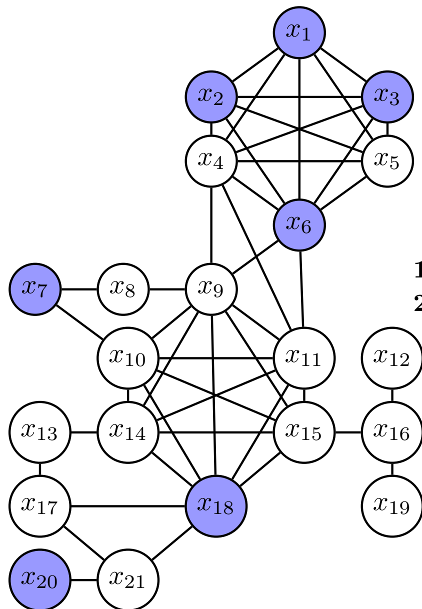
VNS/LDS+CP

(Loudni & Boizumault, EJOR 2008)

Pour k une dimension de voisinage, et s une solution courante :

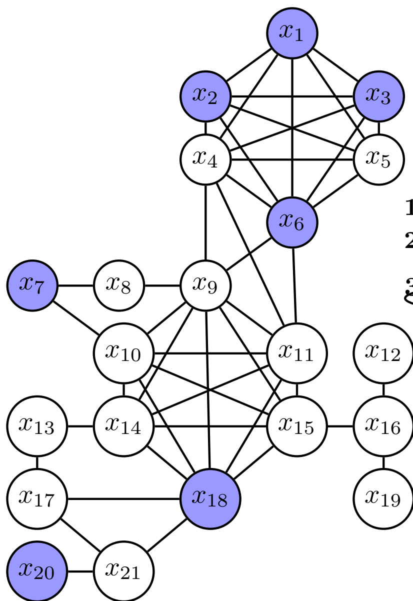
- N_k correspond à l'ensemble des combinaisons de k variables parmi X,
- on sélectionne dans N_k un sous-ensemble de k variables, noté X_{un},
- une affectation partielle A_k est définie en désaffectant les k variables : $A_k = s \setminus \{(x_i = v_i)\ t.q.\ x_i \in X_{un}\}$,
- les variables désaffectées sont ensuite reconstruites par une recherche arborescente partielle (LDS+CP).

VNS/LDS+CP



1. $s = \{(x_1 = v_1), \ldots, (x_{21} = v_{21})\}$

2. $X_{un} = \{x_1, x_2, x_3, x_6, x_7, x_{18}, x_{20}\}$



1. $s = \{(x_1 = v_1), \ldots, (x_{21} = v_{21})\}$

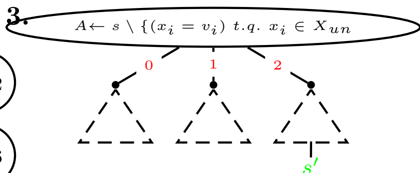
2. $X_{un} = \{x_1, x_2, x_3, x_6, x_7, x_{18}, x_{20}\}$

3. $A \leftarrow s \setminus \{(x_i = v_i)\ t.q.\ x_i \in X_{un}$

1. $s = \{(x_1 = v_1), \ldots, (x_{21} = v_{21})\}$

2. $X_{un} = \{x_1, x_2, x_3, x_6, x_7, x_{18}, x_{20}\}$

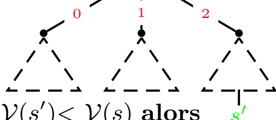
3.



1. $s = \{(x_1 = v_1), \ldots, (x_{21} = v_{21})\}$

2. $X_{un} = \{x_1, x_2, x_3, x_6, x_7, x_{18}, x_{20}\}$

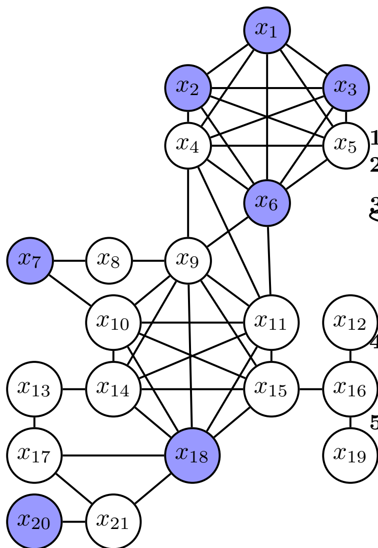
3. $A \leftarrow s \setminus \{(x_i = v_i)\ t.q.\ x_i \in X_{un}$



4. Si $\mathcal{V}(s') < \mathcal{V}(s)$ **alors**

$s \leftarrow s'$ **et** $k \leftarrow k_{min}$

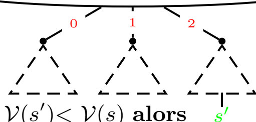
Sinon $k \leftarrow k + 1$



1. $s = \{(x_1 = v_1), \ldots, (x_{21} = v_{21})\}$

2. $X_{un} = \{x_1, x_2, x_3, x_6, x_7, x_{18}, x_{20}\}$

3. $A \leftarrow s \setminus \{(x_i = v_i)\ t.q.\ x_i \in X_{un}$



4. Si $\mathcal{V}(s') < \mathcal{V}(s)$ **alors**

$s \leftarrow s'$ **et** $k \leftarrow k_{min}$

Sinon $k \leftarrow k + 1$

5.

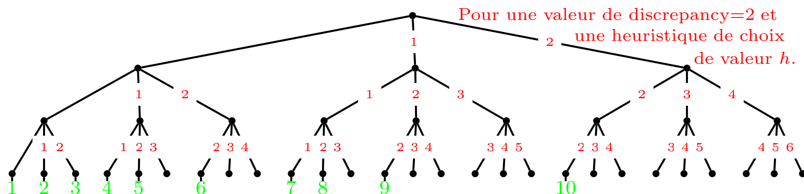
Tant que $(k \leq k_{max})$

Algorithme 16: VNS/LDS+CP

```
function VNS/LDS+CP(X,C, k_init, k_max, δ_max) ;
begin
1     S ← genInitSol() ;
2     k ← k_init;
3     while (k < k_max) ∧ (notTimeOut) do
4         X_un ← Hneighborhood(N_k, S) ;
5         A_k ← S\{(x_i = a) | x_i ∈X_un} ;
6         S' ← LDS+CP(A_k,X_un, δ_max, f(S), S) ;
7         if f(S') < f(S) then
8             S ← S' ;
9             k ← k_init ; // intensification
          end
10        else k ← k + 1 ; // diversification
      end
11    return S ;
end
```

La version n-aire (LDS+CP) étendu à l'optimisation de LDS :

- effectue un parcours en profondeur d'abord,
- effectue seulement l'itération avec la valeur maximale de discrepancy,
- comptabilise une discrepancy de $(k-1)$ lors de la sélection de la k ième valeur d'un domaine.



Exploitation de la structure d'un problème pour guider VNS

Structure d'un problème et décomposition arborescente

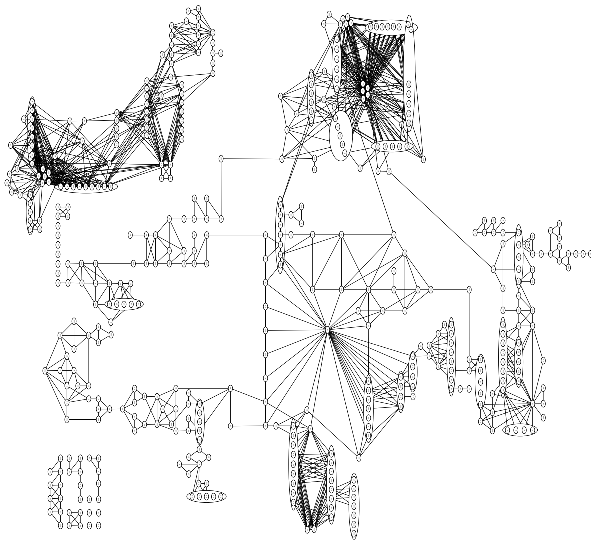
Objectif: Découvrir la **structure** du problème pour l'exploiter.

- trouver la structure des **liens** entre les variables
- rassembler les variables **fortement liées**

➥ Une méthode existante: **la décomposition arborescente**.

Structure de l'instance Scen08 (RLFAP)

$(n = 458, e = 5286)$



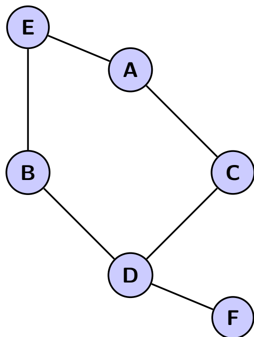
Décomposition arborescente

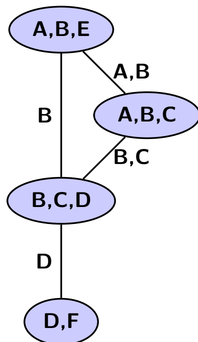
- Introduite par Robertson et Seymour (Journal of algorithms,1986)
- Partitionner un graphe en sous-groupes (clusters) de sommets fortement liés
- Créer un graphe acyclique de clusters

Structure d'un CFN et décomposition arborescente

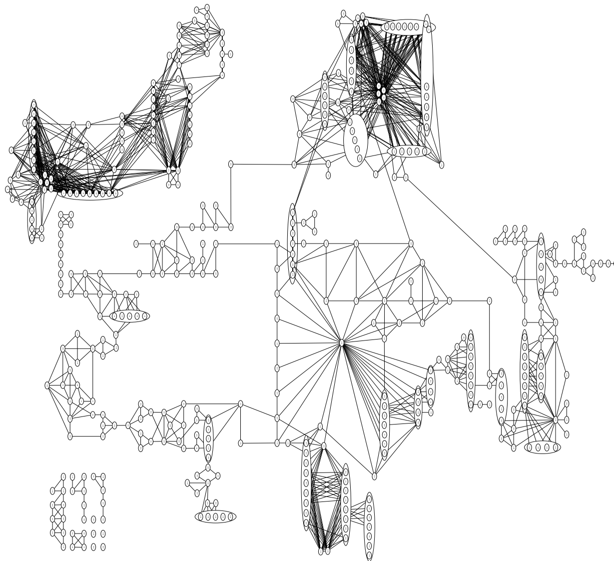
Soit $P = (\mathrm{X},\mathrm{D},\mathrm{W},k)$ un CNF défini de la façon suivante:

- X= $\{A, B, C, D, E, F\}$
- W= $\{w_1(A, B, E), w_2(A, C), w_3(D, C), w_4(B, D), w_5(D, F)\}$





Structure de l'instance Scen08 (RLFAP) ($n = 458, e = 5286$)



Décomposition de l'instance Scen08 (RLFAP)

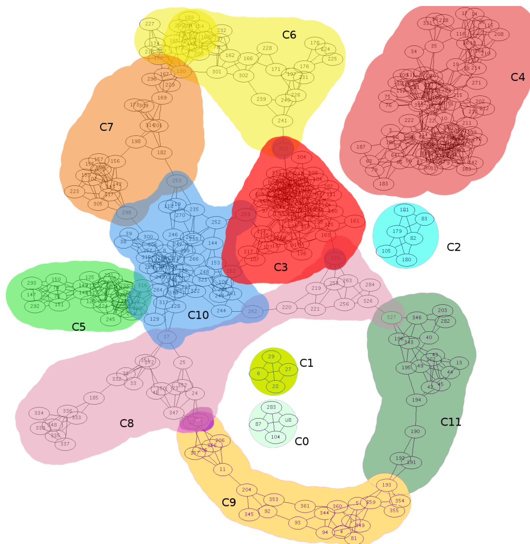


Figure : Décomposition simplifiée de l'instance Scen08 (Simon de Givry)

Motivations

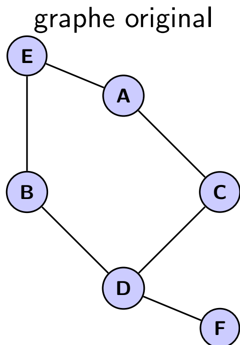
- De nombreux travaux dans le cadre des méthodes de recherche complète: `BTD` (Jégou et Terrioux, AI'03), `RDS-BTD` (Sanchez *et al.*, IJCAI'09)...
- Aucun travail dans le cadre des méthodes de recherche locale

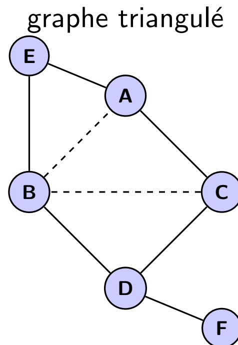
Contributions

- **Un premier schéma générique** d'exploitation de la décomposition arborescente:
  - DGVNS
  - Stratégies d'exploration
- **Deux méthodes de raffinement** de la décomposition arborescente:
  - *Tightness Dependent Tree Decomposition*
  - Raffinement par fusion de cluster
- **Deux schémas** d'exploitation des séparateurs:
  - SGVNS
  - ISGVNS

Décomposition Arborescente

Une méthode approchée basée sur la triangulation:

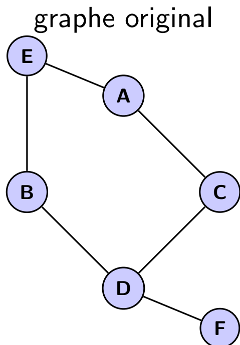


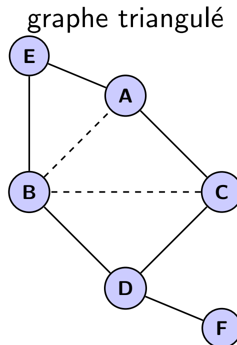


Décomposition Arborescente

Une méthode approchée basée sur la triangulation:

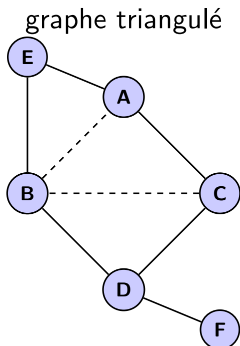
1. Triangulation (MCS),

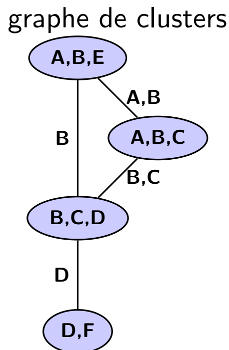




Une méthode approchée basée sur la triangulation:

1. Triangulation (MCS),
2. Recherche des cliques maximales (graphe de clusters),

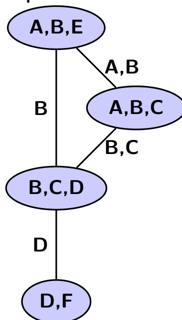


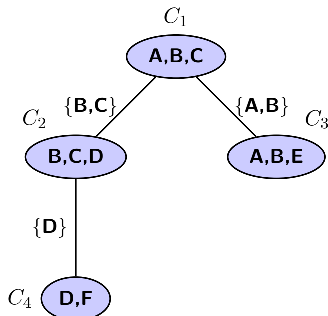


Décomposition Arborescente

Une méthode approchée basée sur la triangulation:

1. Triangulation (MCS),
2. Recherche des cliques maximales (graphe de clusters),
3. Arbre de jointure (arbre couvrant de poids maximal).





Contribution I: Decomposition Guided VNS (Fontaine *et al.*, RAIRO 2013)

Exploiter le graphe de clusters afin de construire des **voisinages plus pertinents** au sein de VNS.

- Le graphe de clusters est une **carte** du problème.
- Les clusters regroupent des variables fortement **corrélées**.
- La couverture de l'intégralité des clusters permet une meilleure **diversification** de la recherche.

Pour favoriser les mouvements dans des régions du problème **fortement liées**, DGVNS utilise des structures de voisinages notées $N_{k,i}$:

Structure de voisinage $N_{k,i}$

Soit G un graphe de contraintes et $G_T=(C_T,E_T)$ un graphe de clusters associé à G. Soit $C_i \in C_T$ un cluster de G_T et k la **dimension de voisinage**. $N_{k,i}$ correspond à toutes les **combinaisons** possibles de k variables dans C_i.

Les voisinages sont obtenus en **désaffectant** k variables dans C_i sélectionnées par une heuristique de choix de variables (Conflict-var).

Decomposition Guided VNS

Soit $\mathbf{N_{k,i}}$ la structure de voisinage courante et **S** la solution courante.

- Centrer la recherche sur un cluster C_i,
- Sélectionner dans C_i un ensemble de k variables à désaffecter dans S,
- Reconstruire la solution S pour obtenir une solution S',
- Changer de voisinage.

Le **changement de voisinage** détermine la manière d'**intensifier** et de **diversifier** la recherche en fonction de la qualité de la nouvelle solution S'.

➥ Trois stratégies d'intensification/diversification pour DGVNS (Loudni *et al.*,HM'13).

DGVNS-1: Changer systématiquement de cluster

Idée: diversifier l'exploration en considérant successivement tous les C_i.

Changement de voisinage pour DGVNS-1

- Si S' est de meilleure qualité que S:
  - **réduire** la dimension de voisinage à k_{init}
  - **diriger** la recherche vers le prochain cluster
- Sinon:
  - **augmenter** la dimension du voisinage
  - **diriger** la recherche vers le prochain cluster

➥ **diversification "agressive"** de la recherche.

DGVNS-2: Changer de cluster en l'absence d'amélioration

Idée: intensifier l'exploration dans le même cluster lors d'une amélioration.

Changement de voisinage pour DGVNS-2

- Si S' est de meilleure qualité que S:
  - **réduire** la dimension de voisinage à k_{init}
  - **continuer** la recherche dans le cluster courant. (Intensification)
- Sinon:
  - **augmenter** la dimension du voisinage
  - **diriger** la recherche vers le prochain cluster.(Diversification)

➥ Favoriser **l'équilibre** entre intensification et diversification.

DGVNS-3: Changer de cluster après chaque amélioration

Idée: rester dans un cluster tant qu'aucune amélioration n'est obtenue.

Changement de voisinage pour DGVNS-3

- Si S' est de meilleure qualité que S:
  - **réduire** la dimension de voisinage à k_{init}
  - **diriger** la recherche vers le prochain cluster
- Sinon:
  - **augmenter** la dimension du voisinage
  - **continuer** la recherche dans le cluster courant

➥ Étudier l'intérêt du changement de cluster après une amélioration

Expérimentations

Implémentation

- Toulbar2
  - ➥ `https://mulcyber.toulouse.inra.fr/projects/toulbar2/`
- C++

Méthodes comparées

- `VNS/LDS+CP` (Loudni *et al.*, EJOR'08)
- `IDWalk` (Neveu *et al.*, CP'04)
- `DGVNS-1`

Protocole expérimental

- 50 runs par instance,
- timeout: 1, 2, 4 heures par run.
- `AMD Opteron 2,6 GHz, 256 Go de RAM`

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-1	49/50	117	6228
$n = 200,\ e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
#412	DGVNS-**1**	**36/50**	**84**	**32381**
$n = 300,\ e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	32381
#414	DGVNS-**1**	**38/50**	**554**	**38478**
$n = 364, e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
#505	DGVNS-**1**	**50/50**	**63**	**21253**
$n = 240,\ e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
#507	DGVNS-**1**	**33/50**	**71**	**27390**
$n = 311,\ e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
#509	DGVNS-**1**	**40/50**	**265**	**36446**
$n = 348,\ e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-1	49/50	117	6228
$n = 200$, $e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
#412	DGVNS-**1**	**36/50**	**84**	**32381**
$n = 300$, $e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	32381
#414	DGVNS-**1**	**38/50**	**554**	**38478**
$n = 364, e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
#505	DGVNS-**1**	**50/50**	**63**	**21253**
$n = 240$, $e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
#507	DGVNS-**1**	**33/50**	**71**	**27390**
$n = 311$, $e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
#509	DGVNS-**1**	**40/50**	**265**	**36446**
$n = 348$, $e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-1	49/50	117	6228
$n = 200,\ e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
#412	DGVNS-**1**	**36/50**	**84**	**32381**
$n = 300,\ e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	32381
#414	DGVNS-**1**	**38/50**	**554**	**38478**
$n = 364, e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
#505	DGVNS-**1**	**50/50**	**63**	**21253**
$n = 240,\ e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
#507	DGVNS-**1**	**33/50**	**71**	**27390**
$n = 311,\ e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
#509	DGVNS-**1**	**40/50**	**265**	**36446**
$n = 348,\ e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408 $n = 200,\ e = 2232$ $S^* = 6228$	DGVNS-1	49/50	117	6228
	VNS/LDS+CP	26/50	149	6228
	ID-Walk	**50/50**	**3**	**6228**
#412 $n = 300,\ e = 4348$ $S^* = 32381$	DGVNS-**1**	**36/50**	**84**	**32381**
	VNS/LDS+CP	32/50	130	32381
	ID-Walk	10/50	2102	32381
#414 $n = 364, e = 10108$ $S^* = 38478$	DGVNS-**1**	**38/50**	**554**	**38478**
	VNS/LDS+CP	12/50	434	38481
	ID-Walk	0/50	-	38481
#505 $n = 240,\ e = 2242$ $S^* = 21253$	DGVNS-**1**	**50/50**	**63**	**21253**
	VNS/LDS+CP	41/50	143	21253
	ID-Walk	**50/50**	**358**	**21253**
#507 $n = 311,\ e = 5732$ $S^* = 27390$	DGVNS-**1**	**33/50**	**71**	**27390**
	VNS/LDS+CP	11/50	232	27391
	ID-Walk	7/50	1862	27391
#509 $n = 348,\ e = 8624$ $S^* = 36446$	DGVNS-**1**	**40/50**	**265**	**36446**
	VNS/LDS+CP	12/50	598	36448
	ID-Walk	0/50	-	36450

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-1	49/50	117	6228
$n = 200,\ e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
#412	DGVNS-**1**	**36/50**	**84**	**32381**
$n = 300,\ e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	32381
#414	DGVNS-**1**	**38/50**	**554**	**38478**
$n = 364, e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
#505	DGVNS-**1**	**50/50**	**63**	**21253**
$n = 240,\ e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
#507	DGVNS-**1**	**33/50**	**71**	**27390**
$n = 311,\ e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
#509	DGVNS-**1**	**40/50**	**265**	**36446**
$n = 348,\ e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-1	49/50	117	6228
$n = 200$, $e = 2232$	VNS/LDS+CP	26/50	149	6228
$S^* = 6228$	ID-Walk	**50/50**	**3**	**6228**
#412	DGVNS-1	**36/50**	**84**	**32381**
$n = 300$, $e = 4348$	VNS/LDS+CP	32/50	130	32381
$S^* = 32381$	ID-Walk	10/50	2102	32381
#414	DGVNS-1	**38/50**	**554**	**38478**
$n = 364, e = 10108$	VNS/LDS+CP	12/50	434	38481
$S^* = 38478$	ID-Walk	0/50	-	38481
#505	DGVNS-1	**50/50**	**63**	**21253**
$n = 240$, $e = 2242$	VNS/LDS+CP	41/50	143	21253
$S^* = 21253$	ID-Walk	**50/50**	**358**	**21253**
#507	DGVNS-1	**33/50**	**71**	**27390**
$n = 311$, $e = 5732$	VNS/LDS+CP	11/50	232	27391
$S^* = 27390$	ID-Walk	7/50	1862	27391
#509	DGVNS-1	**40/50**	**265**	**36446**
$n = 348$, $e = 8624$	VNS/LDS+CP	12/50	598	36448
$S^* = 36446$	ID-Walk	0/50	-	36450

Problèmes traités:

- `RLFAP`: 3 instances;
- `GRAPH`: 4 instances;
- `SPOT5`: 6 instances;
- `tagSNP`: 13 instances.

Profils de rapport de performance (Dolan et Moré, 2002)

Pour une méthode $a \in A$, pour chaque instance $p \in P$, on évalue les performances de a sur p selon une **mesure** notée $M_{a,p}$:

- **temps de calcul** (pénalisé) : $t_{a,p}$
- **taux de succès**: $s_{a,p}$

Profils de rapport de performance (2/3)

Rapport de performance pour une mesure $M_{a,p}$:

$$r_{a,p} = \frac{M_{a,p}}{min(\{M_{a_k,p} | a_k \in A\})}$$

- Si $\mathbf{r_{a,p} = 1}$, alors la méthode a obtient les meilleurs résultats sur p
- Si $\mathbf{r_{a,p}} = \alpha$, alors, pour p, la meilleure méthode est α fois plus performante que a

Profils de rapport de performance (3/3)

Performance fractionnaire:

$$\rho_a(\tau) = \frac{|\{p | r_{a,p} \leq \tau\}|}{|P|}$$

- $\boldsymbol{\rho_a(1)}$: proportion de problèmes sur lesquels a obtient les meilleures performances
- plus **l'aire sous la courbe de ρ_a** est grande, plus a est performante

Profils de rapport de performance: Mesure: Temps de calcul

- aire sous la courbe
  - DGVNS-1: 1
  - VNS/LDS+CP: 0.875
- $\rho(1)$
  - DGVNS-1: 0.884
  - VNS/LDS+CP: 0.192

Profils de rapport de performance: Mesure: Taux de succès

- aire sous la courbe
  - `DGVNS-1`: 1
  - `VNS/LDS+CP`: 0.812
- $\rho(1)$
  - `DGVNS-1`: 0.807
  - `VNS/LDS+CP`: 0.269

Expérimentations

Implémentation

- Toulbar2
- C++

Méthodes comparées

- DGVNS-1
- DGVNS-2
- DGVNS-3

Protocole expérimental

- 50 runs par instance,
- timeout: 1, 2, 4 heures par run.
- AMD Opteron 2,6 GHz, 256 Go de RAM

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-3	50/50	1697	6228
$n = 200,\ e = 2232$	DGVNS-2	**50/50**	**81**	**6228**
$S^* = 6228$	DGVNS-1	49/50	117	6228
#412	DGVNS-3	0/50	-	32384 (32383)
$n = 300,\ e = 4348$	DGVNS-2	**48/50**	**484**	**32381**
$S^* = 32381$	DGVNS-1	36/50	84	32381
#414	DGVNS-3	0/50	-	38486 (38482)
$n = 364, e = 10108$	DGVNS-2	**45/50**	**670**	**38478**
$S^* = 38478$	DGVNS-1	38/50	554	38478
#505	DGVNS-3	36/50	3094	21253
$n = 240,\ e = 2242$	DGVNS-2	50/50	90	21253
$S^* = 21253$	DGVNS-1	**50/50**	**63**	**21253**
#507	DGVNS-3	0/50	-	27396 (27393)
$n = 311,\ e = 5732$	DGVNS-2	**45/50**	**463**	**27390**
$S^* = 27390$	DGVNS-1	33/50	71	27390
#509	DGVNS-3	0/50	-	36454 (36450)
$n = 348,\ e = 8624$	DGVNS-2	**47/50**	**509**	**36446**
$S^* = 36446$	DGVNS-1	40/50	265	36446

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-3	50/50	1697	6228
$n = 200,\ e = 2232$	DGVNS-2	**50/50**	**81**	**6228**
$S^* = 6228$	DGVNS-1	49/50	117	6228
#412	DGVNS-3	0/50	-	32384 (32383)
$n = 300,\ e = 4348$	DGVNS-2	**48/50**	**484**	**32381**
$S^* = 32381$	DGVNS-1	36/50	84	32381
#414	DGVNS-3	0/50	-	38486 (38482)
$n = 364, e = 10108$	DGVNS-2	**45/50**	**670**	**38478**
$S^* = 38478$	DGVNS-1	38/50	554	38478
#505	DGVNS-3	36/50	3094	21253
$n = 240,\ e = 2242$	DGVNS-2	50/50	90	21253
$S^* = 21253$	DGVNS-1	**50/50**	**63**	**21253**
#507	DGVNS-3	0/50	-	27396 (27393)
$n = 311,\ e = 5732$	DGVNS-2	**45/50**	**463**	**27390**
$S^* = 27390$	DGVNS-1	33/50	71	27390
#509	DGVNS-3	0/50	-	36454 (36450)
$n = 348,\ e = 8624$	DGVNS-2	**47/50**	**509**	**36446**
$S^* = 36446$	DGVNS-1	40/50	265	36446

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-3	50/50	1697	6228
$n = 200$, $e = 2232$	DGVNS-2	**50/50**	**81**	**6228**
$S^* = 6228$	DGVNS-1	49/50	117	6228
#412	DGVNS-3	0/50	-	32384 (32383)
$n = 300$, $e = 4348$	DGVNS-2	**48/50**	**484**	**32381**
$S^* = 32381$	DGVNS-1	36/50	84	32381
#414	DGVNS-3	0/50	-	38486 (38482)
$n = 364, e = 10108$	DGVNS-2	**45/50**	**670**	**38478**
$S^* = 38478$	DGVNS-1	38/50	554	38478
#505	DGVNS-3	36/50	3094	21253
$n = 240$, $e = 2242$	DGVNS-2	50/50	90	21253
$S^* = 21253$	DGVNS-1	**50/50**	**63**	**21253**
#507	DGVNS-3	0/50	-	27396 (27393)
$n = 311$, $e = 5732$	DGVNS-2	**45/50**	**463**	**27390**
$S^* = 27390$	DGVNS-1	33/50	71	27390
#509	DGVNS-3	0/50	-	36454 (36450)
$n = 348$, $e = 8624$	DGVNS-2	**47/50**	**509**	**36446**
$S^* = 36446$	DGVNS-1	40/50	265	36446

Résultats sur les instances SPOT5

Instance	Méthode	Succ.	Temps (s.)	Moy.
#408	DGVNS-3	50/50	1697	6228
$n = 200,\ e = 2232$	DGVNS-2	**50/50**	**81**	**6228**
$S^* = 6228$	DGVNS-1	49/50	117	6228
#412	DGVNS-3	0/50	-	32384 (32383)
$n = 300,\ e = 4348$	DGVNS-2	**48/50**	**484**	**32381**
$S^* = 32381$	DGVNS-1	36/50	84	32381
#414	DGVNS-3	0/50	-	38486 (38482)
$n = 364, e = 10108$	DGVNS-2	**45/50**	**670**	**38478**
$S^* = 38478$	DGVNS-1	38/50	554	38478
#505	DGVNS-3	36/50	3094	21253
$n = 240,\ e = 2242$	DGVNS-2	50/50	90	21253
$S^* = 21253$	DGVNS-1	**50/50**	**63**	**21253**
#507	DGVNS-3	0/50	-	27396 (27393)
$n = 311,\ e = 5732$	DGVNS-2	**45/50**	**463**	**27390**
$S^* = 27390$	DGVNS-1	33/50	71	27390
#509	DGVNS-3	0/50	-	36454 (36450)
$n = 348,\ e = 8624$	DGVNS-2	**47/50**	**509**	**36446**
$S^* = 36446$	DGVNS-1	40/50	265	36446

Profils de rapport de performance: Temps de calcul

- aire sous la courbe
  - `DGVNS-1`: 1
  - `DGVNS-2`: 0.952
- $\rho(1)$
  - `DGVNS-1`: 0.65
  - `DGVNS-2`: 0.5

Profils de rapport de performance: Taux de succès

- aire sous la courbe
  - `DGVNS-1`: 0.935
  - `DGVNS-2`: 1
- $\rho(1)$
  - `DGVNS-1`: 0.5
  - `DGVNS-2`: 0.731

Bilan sur les expérimentations (Contribution I)

- DGVNS **surclasse** VNS/LDS+CP et .
- DGVNS-1 obtient les meilleurs **temps de calculs**.
- DGVNS-2 obtient les meilleurs **taux de succès**.

Contribution II

- Un premier schéma générique d'exploitation de la décomposition arborescente:
  - `DGVNS`
  - Stratégies d'exploration
- Deux méthodes de raffinement de la décomposition arborescente:
  - *Tightness Dependent Tree Decomposition*
  - Raffinement par fusion de cluster
- Deux schémas d'exploitation des séparateurs:
  - `SGVNS`
  - `ISGVNS`

Contribution II

- Un premier schéma générique d'exploitation de la décomposition arborescente:
  - DGVNS
  - Stratégies d'exploration
- Deux méthodes de raffinement de la décomposition arborescente:
  - *Tightness Dependent Tree Decomposition*
  - Raffinement par fusion de cluster
- Deux schémas d'exploitation des séparateurs:
  - SGVNS
  - ISGVNS

Constats

- Intérêt de la décomposition arborescente pour guider `VNS`
- Sous sa forme brute, 2 principaux inconvénients:
  - Pas de prise en compte des **caractéristiques** des fonctions de coût (dureté, répartition des coûts...).
  - Fort **recouvrement** entre les clusters.

Constats

- Intérêt de la décomposition arborescente pour guider `VNS`
- Sous sa forme brute, 2 principaux inconvénients:
  - Pas de prise en compte des **caractéristiques** des fonctions de coût (dureté, répartition des coûts...): `TDTD`
  - Fort **recouvrement** entre les clusters.

Idée

Orienter la recherche vers les sous-parties du problème les plus dures à résoudre.

Principe

Lors de la décomposition, ignorer les fonctions de coût de faible dureté.

Résultats

- Apports très significatifs sur les instances des problèmes `RLFAP`, `GRAPH` et `SPOT5`
- Non pertinent pour les instances du problème `tagSNP`

Constats

- Intérêt de la décomposition arborescente pour guider VNS
- Sous sa forme brute, 2 principaux inconvénients:
  - Pas de prise en compte des **caractéristiques** des fonctions de coût (dureté, répartition des coûts...): TDTD
  - Fort **recouvrement** entre les clusters.

Constats

- Intérêt de la décomposition arborescente pour guider `VNS`
- Sous sa forme brute, 2 principaux inconvénients:
  - Pas de prise en compte des **caractéristiques** des fonctions de coût (dureté, répartition des coûts...): `TDTD`
  - Fort **recouvrement** entre les clusters: Fusion

Raffinement par fusion de clusters (Fontaine *et al.*, RAIRO'13)

- Le recouvrement entre clusters **appauvrit** la diversification de DGVNS.
- La taille du séparateur entre deux clusters doit être **très petite** par rapport à la taille de ces clusters.

Deux critères de fusion:

- s_{max}: fusion basée sur **la taille du séparateur**
- Ab_{max}: fusion basée sur **l'absorption**

Definition

Soit C_i et C_j deux clusters. L'absorption de C_i par C_j est définie par:

$$Ab(C_i, C_j) = \frac{|Sep(C_i, C_j)|}{|C_i|}$$

Expérimentations

Implémentation

- Toulbar2
- C++

Méthodes comparées

- DGVNS-1
- DGVNS-1 + s_{max}: $s_{max} \in \{4, 8, 12, 16, 24, 32\}$
- DGVNS-1 + Ab_{max}: $Ab_{max} = 70\%$.

Protocole expérimental

- 50 runs par instance,
- timeout: 1,2,4 heures par run.
- AMD Opteron 2,6 GHz, 256 Go de RAM

Instance	Méthode	Succ.	Temps (s.)	Moy.
Scen06	DGVNS-1	50/50	112	3389
$n = 100, e = 1222$	$s_{max} = 4$	50/50	368	3389
$S^* = 3389$	DGVNS-Ab_{max}	**50/50**	**103**	**3389**
Scen07	DGVNS-1	40/50	317	345614
$n = 200,\ e = 2665$	$s_{max} = 8$	8/50	2379	343803
$S^* = 343592$	DGVNS-Ab_{max}	**50/50**	**514**	**343592**
Scen08	DGVNS-1	3/50	1811	275
$n = 458,\ e = 5286$	$s_{max} = 16$	0/50	-	312 (309)
$S^* = 262$	DGVNS-Ab_{max}	**5/50**	**726**	**274**
graph05	DGVNS-1	**50/50**	**10**	**221**
$n = 100,\ e = 1034$	$s_{max} = 24$	0/50	-	1654 (1654)
$S^* = 221$	DGVNS-Ab_{max}	50/50	17	221
graph06	DGVNS-1	50/50	367	4123
$n = 200, e = 1970$	$s_{max} = 32$	**50/50**	**230**	**4123**
$S^* = 4123$	DGVNS-Ab_{max}	**50/50**	**260**	**4123**
graph11	DGVNS-1	8/50	3046	4234
$n = 340, e = 3417$	$s_{max} = 32$	0/50	-	23144 (23085)
$S^* = 3080$	DGVNS-Ab_{max}	**25/50**	**2967**	**3789**
graph13	DGVNS-1	0/50	-	22489 (18639)
$n = 458, e = 4915$	$s_{max} =$**32**	**7/50**	**3247**	**11766**
$S^* = 10110$	DGVNS-Ab_{max}	**1/50**	**3520**	**16756**

Résultats sur les instances GRAPH et RLFAP

Instance	Méthode	Succ.	Temps (s.)	Moy.
Scen06	DGVNS-1	50/50	112	3389
$n = 100, e = 1222$	$s_{max} = 4$	50/50	368	3389
$S^* = 3389$	DGVNS-Ab_{max}	**50/50**	**103**	**3389**
Scen07	DGVNS-1	40/50	317	345614
$n = 200, e = 2665$	$s_{max} = 8$	8/50	2379	343803
$S^* = 343592$	DGVNS-Ab_{max}	**50/50**	**514**	**343592**
Scen08	DGVNS-1	3/50	1811	275
$n = 458, e = 5286$	$s_{max} = 16$	0/50	-	312 (309)
$S^* = 262$	DGVNS-Ab_{max}	**5/50**	**726**	**274**
graph05	DGVNS-1	**50/50**	**10**	**221**
$n = 100, e = 1034$	$s_{max} = 24$	0/50	-	1654 (1654)
$S^* = 221$	DGVNS-Ab_{max}	50/50	17	221
graph06	DGVNS-1	50/50	367	4123
$n = 200, e = 1970$	$s_{max} = 32$	**50/50**	**230**	**4123**
$S^* = 4123$	DGVNS-Ab_{max}	**50/50**	**260**	**4123**
graph11	DGVNS-1	8/50	3046	4234
$n = 340, e = 3417$	$s_{max} = 32$	0/50	-	23144 (23085)
$S^* = 3080$	DGVNS-Ab_{max}	**25/50**	**2967**	**3789**
graph13	DGVNS-1	0/50	-	22489 (18639)
$n = 458, e = 4915$	$s_{max} =$ **32**	**7/50**	**3247**	**11766**
$S^* = 10110$	DGVNS-Ab_{max}	**1/50**	**3520**	**16756**

Résultats sur les instances GRAPH et RLFAP

Instance	Méthode	Succ.	Temps (s.)	Moy.
Scen06	DGVNS-1	50/50	112	3389
$n = 100, e = 1222$	$s_{max} = 4$	50/50	368	3389
$S^* = 3389$	DGVNS-Ab_{max}	**50/50**	**103**	**3389**
Scen07	DGVNS-1	40/50	317	345614
$n = 200, e = 2665$	$s_{max} = 8$	8/50	2379	343803
$S^* = 343592$	DGVNS-Ab_{max}	**50/50**	**514**	**343592**
Scen08	DGVNS-1	3/50	1811	275
$n = 458, e = 5286$	$s_{max} = 16$	0/50	-	312 (309)
$S^* = 262$	DGVNS-Ab_{max}	**5/50**	**726**	**274**
graph05	DGVNS-1	**50/50**	**10**	**221**
$n = 100, e = 1034$	$s_{max} = 24$	0/50	-	1654 (1654)
$S^* = 221$	DGVNS-Ab_{max}	50/50	17	221
graph06	DGVNS-1	50/50	367	4123
$n = 200, e = 1970$	$s_{max} = 32$	**50/50**	**230**	**4123**
$S^* = 4123$	DGVNS-Ab_{max}	**50/50**	**260**	**4123**
graph11	DGVNS-1	8/50	3046	4234
$n = 340, e = 3417$	$s_{max} = 32$	0/50	-	23144 (23085)
$S^* = 3080$	DGVNS-Ab_{max}	**25/50**	**2967**	**3789**
graph13	DGVNS-1	0/50	-	22489 (18639)
$n = 458, e = 4915$	$s_{max} =$ **32**	**7/50**	**3247**	**11766**
$S^* = 10110$	DGVNS-Ab_{max}	**1/50**	**3520**	**16756**

Résultats sur les instances GRAPH et RLFAP

Instance	Méthode	Succ.	Temps (s.)	Moy.
Scen06	DGVNS-1	50/50	112	3389
$n = 100, e = 1222$	$s_{max} = 4$	50/50	368	3389
$S^* = 3389$	DGVNS-Ab_{max}	**50/50**	**103**	**3389**
Scen07	DGVNS-1	40/50	317	345614
$n = 200,\ e = 2665$	$s_{max} = 8$	8/50	2379	343803
$S^* = 343592$	DGVNS-Ab_{max}	**50/50**	**514**	**343592**
Scen08	DGVNS-1	3/50	1811	275
$n = 458,\ e = 5286$	$s_{max} = 16$	0/50	-	312 (309)
$S^* = 262$	DGVNS-Ab_{max}	**5/50**	**726**	**274**
graph05	DGVNS-1	**50/50**	**10**	**221**
$n = 100,\ e = 1034$	$s_{max} = 24$	0/50	-	1654 (1654)
$S^* = 221$	DGVNS-Ab_{max}	50/50	17	221
graph06	DGVNS-1	50/50	367	4123
$n = 200, e = 1970$	$s_{max} = 32$	**50/50**	**230**	**4123**
$S^* = 4123$	DGVNS-Ab_{max}	**50/50**	**260**	**4123**
graph11	DGVNS-1	8/50	3046	4234
$n = 340, e = 3417$	$s_{max} = 32$	0/50	-	23144 (23085)
$S^* = 3080$	DGVNS-Ab_{max}	**25/50**	**2967**	**3789**
graph13	DGVNS-1	0/50	-	22489 (18639)
$n = 458, e = 4915$	$s_{max} =$ **32**	**7/50**	**3247**	**11766**
$S^* = 10110$	DGVNS-Ab_{max}	**1/50**	**3520**	**16756**

Bilan sur les expérimentations (Contribution II)

1. La fusion des clusters permet **d'améliorer les performances** de DGVNS
2. Réduire la redondance entre clusters **renforce** la diversification de DGVNS
3. Le critère **d'absorption** donne les meilleurs résultats

Contribution III

- Un premier schéma générique d'exploitation de la décomposition arborescente:
  - `DGVNS`,
  - Stratégies d'exploration.
- Deux méthodes de raffinement de la décomposition arborescente:
  - *Tightness Dependent Tree Decomposition*,
  - Raffinement par fusion de cluster.
- Deux schémas d'exploitation des séparateurs.
  - `SGVNS`
  - `ISGVNS`

Contribution III

- Un premier schéma générique d'exploitation de la décomposition arborescente:
  - `DGVNS`,
  - Stratégies d'exploration.
- Deux méthodes de raffinement de la décomposition arborescente:
  - *Tightness Dependent Tree Decomposition*,
  - Raffinement par fusion de cluster.
- Deux schémas d'exploitation des séparateurs.
  - `SGVNS`
  - `ISGVNS`

Contribution III

Constats

- Les séparateurs constituent des points clefs du problème,
- Leur réaffectation a un impact sur les clusters qui les contiennent.

Objectif

Exploitation des clusters et des séparateurs pour guider VNS

Contributions

- SGVNS: utilisation des séparateurs pour construire des voisinages,
- ISGVNS: propagation des nouvelles affectations à travers les séparateurs.

Contribution III

Constats

- Les séparateurs constituent des points clefs du problème,
- Leur réaffectation a un impact sur les clusters qui les contiennent.

Objectif

Exploitation des clusters et des séparateurs pour guider VNS

Contributions

- SGVNS: utilisation des séparateurs pour construire des voisinages,
- ISGVNS: propagation des nouvelles affectations à travers les séparateurs.

Contribution: Intensified Separators Guided VNS (Loudni *et al.*, ENDM'12)

- Basée sur le même schéma que DGVNS.
- Exploitation des variables ayant mené à une amélioration.
  - identification des clusters contenant les variables modifiées,
  - orientation de la recherche vers ces clusters.

Intensification/Diversification pour ISGVNS

Soit S' une nouvelle solution de coût inférieur à la solution courante S. Nous utiliserons les notations suivantes:

- V_c : l'ensemble des variables réassignées dans S',
- C_w : l'ensemble des clusters contenant au moins une variable de V_c.

Le changement de voisinage pour ISGVNS s'exécute de la façon suivante:

- Si la solution est améliorée, intensifier la recherche:
  - calculer V_c et C_w
  - ajouter C_w à la fin de P_{List}
  - ajouter les variables de V_c à la liste taboue pour $|P_{list}|$ iterations
  - réduire k à k_{init} et se déplacer sur le cluster suivant:
    - si $P_{list} \neq \emptyset$: le prochain cluster de P_{list}
    - sinon $C_{succ(i)}$
- Sinon, diversifier en incrémentant k et en se déplaçant sur le cluster suivant:
  - si $P_{list} \neq \emptyset$: le prochain cluster de P_{list}
  - sinon $C_{succ(i)}$

Expérimentations

Implémentation

- Toulbar2
- C++

Méthodes comparées

- ISGVNS
- DGVNS-1

Protocole expérimental

- 50 runs par instance,
- timeout: 1, 2, 4 heures par run.
- AMD Opteron 2,6 GHz, 256 Go de RAM

Résultats sur les instances tagSNP

Instance	Method	Succ.	Time	Avg
#3792 $n = 528$, $e = 12084$	DGVNS-1	50/50	954	6359805
$S^* = 6359805$	ISGVNS	**50/50**	**853**	**6359805**
#4449 $n = 464, e = 12540$	DGVNS-1	**50/50**	**665**	**5094256**
$S^* = 5094256$	ISGVNS	50/50	675	5094256
#6835 $n = 496, e = 12540$	DGVNS-1	**50/50**	**2409**	**4571108**
$S^* = 4571108$	ISGVNS	50/50	3452	4571108
#8956 $n = 486$, $e = 20832$	DGVNS-1	50/50	4911	6660308
$S^* = 6660308$	ISGVNS	**50/50**	**4118**	**6660308**
#9319 $n = 486$, $e = 20832$	DGVNS-1	50/50	788	6477229
$S^* = 6660308$	ISGVNS	**50/50**	**672**	**6477229**
#16706 $n = 438, e = 6321$	DGVNS-1	50/50	153	2632310
$S^* = 2632310$	ISGVNS	**50/50**	**89**	**2632310**
#14226 $n = 1058$, $e = 47967$	DGVNS-1	46/50	7606	25688751
$S^* = 38318224$	ISGVNS	**50/50**	**9596**	**25665437**

Instance	Method	Succ.	Time	Avg
#3792 $n = 528,\ e = 12084$ $S^* = 6359805$	DGVNS-1	50/50	954	6359805
	ISGVNS	**50/50**	**853**	**6359805**
#4449 $n = 464, e = 12540$ $S^* = 5094256$	DGVNS-1	**50/50**	**665**	**5094256**
	ISGVNS	50/50	675	5094256
#6835 $n = 496, e = 12540$ $S^* = 4571108$	DGVNS-1	**50/50**	**2409**	**4571108**
	ISGVNS	50/50	3452	4571108
#8956 $n = 486,\ e = 20832$ $S^* = 6660308$	DGVNS-1	50/50	4911	6660308
	ISGVNS	**50/50**	**4118**	**6660308**
#9319 $n = 486,\ e = 20832$ $S^* = 6660308$	DGVNS-1	50/50	788	6477229
	ISGVNS	**50/50**	**672**	**6477229**
#16706 $n = 438, e = 6321$ $S^* = 2632310$	DGVNS-1	50/50	153	2632310
	ISGVNS	**50/50**	**89**	**2632310**
#14226 $n = 1058,\ e = 47967$ $S^* = 38318224$	DGVNS-1	46/50	7606	25688751
	ISGVNS	**50/50**	**9596**	**25665437**

Résultats sur les instances tagSNP

Instance	Method	Succ.	Time	Avg
#3792 $n = 528,\ e = 12084$	DGVNS-1	50/50	954	6359805
$S^* = 6359805$	ISGVNS	**50/50**	**853**	**6359805**
#4449 $n = 464, e = 12540$	DGVNS-1	**50/50**	**665**	**5094256**
$S^* = 5094256$	ISGVNS	50/50	675	5094256
#6835 $n = 496, e = 12540$	DGVNS-1	**50/50**	**2409**	**4571108**
$S^* = 4571108$	ISGVNS	50/50	3452	4571108
#8956 $n = 486,\ e = 20832$	DGVNS-1	50/50	4911	6660308
$S^* = 6660308$	ISGVNS	**50/50**	**4118**	**6660308**
#9319 $n = 486,\ e = 20832$	DGVNS-1	50/50	788	6477229
$S^* = 6660308$	ISGVNS	**50/50**	**672**	**6477229**
#16706 $n = 438, e = 6321$	DGVNS-1	50/50	153	2632310
$S^* = 2632310$	ISGVNS	**50/50**	**89**	**2632310**
#14226 $n = 1058,\ e = 47967$	DGVNS-1	46/50	7606	25688751
$S^* = 38318224$	ISGVNS	**50/50**	**9596**	**25665437**

Bilan sur les expérimentations (Contribution III)

1. Exploitation des **séparateurs** pour guider efficacement VNS
2. Deux nouveaux schémas, **ISGVNS et SGVNS**
3. ISGVNS **plus performant** que DGVNS