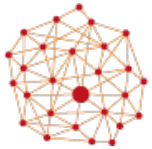


LocalSolver

Modélisation ensembliste avec LocalSolver 7.0

Bertrand Estellon

www.localsolver.com

Qui sommes-nous ?

Bouygues
http://www.bouygues.com

LocalSolver

Solveur mathématique
http://www.localsolver.com

en collaboration avec

LocalSolver

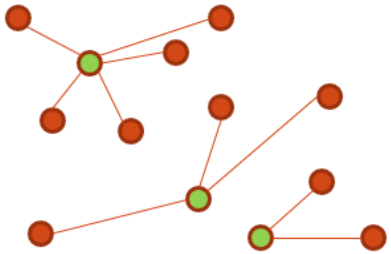
Hybrid math optimization solver

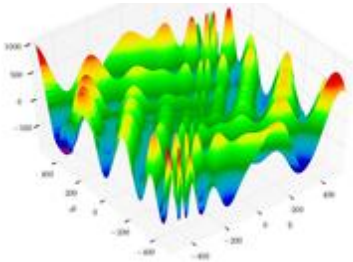
For combinatorial, numerical,
or mixed-variable optimization

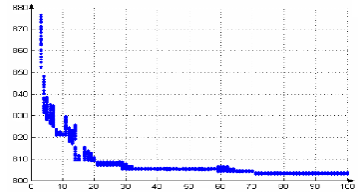
Suited for large-scale
non-convex optimization

Quality solutions in seconds
without tuning

LocalSolver =
LS + CP/SAT + LP/MIP + NLP





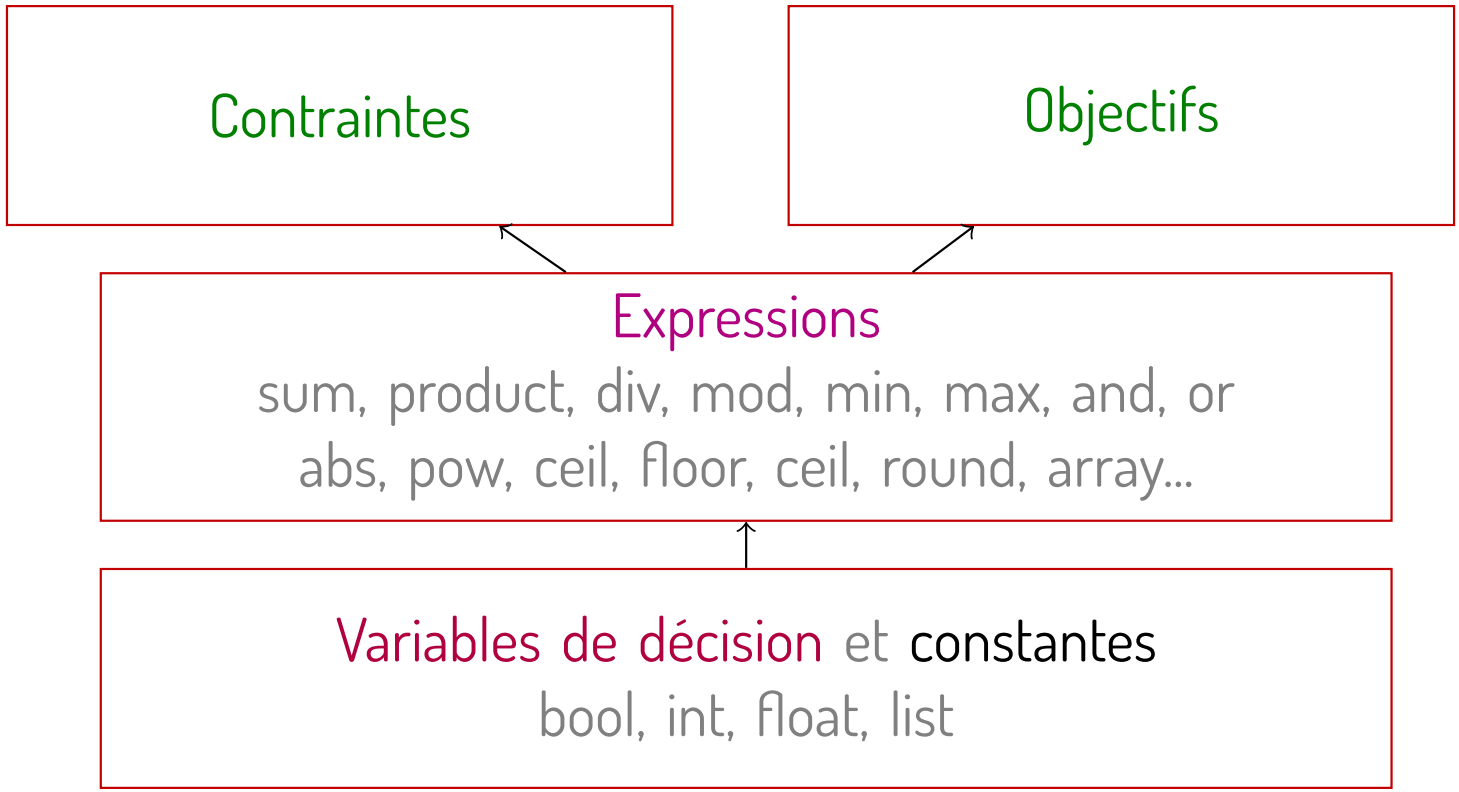


free for academics

www.localsolver.com

LocalSolver

```
function model() {
    x[i in 0..nbItems-1] <- bool();
    knapsackWeight <- sum[i in 0..nbItems-1](weights[i] * x[i]);
    constraint knapsackWeight <= knapsackBound;
    knapsackValue <- sum[i in 0..nbItems-1](prices[i] * x[i]);
    maximize knapsackValue;
}
```

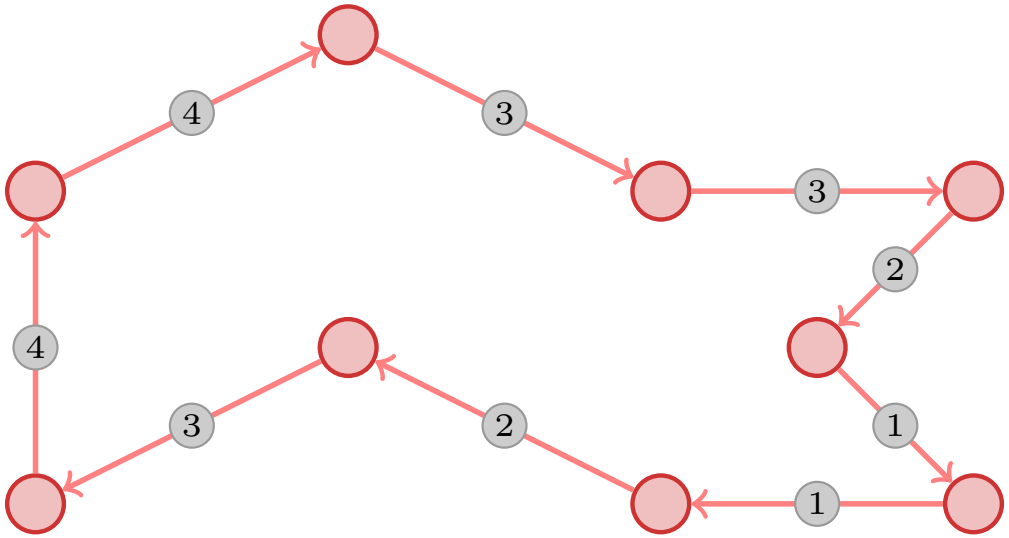


Motivations

Modélisation des problèmes de tournées de véhicules

Voyageur de commerce (TSP)

TSP : Étant donné un ensemble de n villes et les distances qui les séparent, trouver la tournée la plus courte qui visite toutes les villes exactement une fois.



minimiser $\sum$ ●

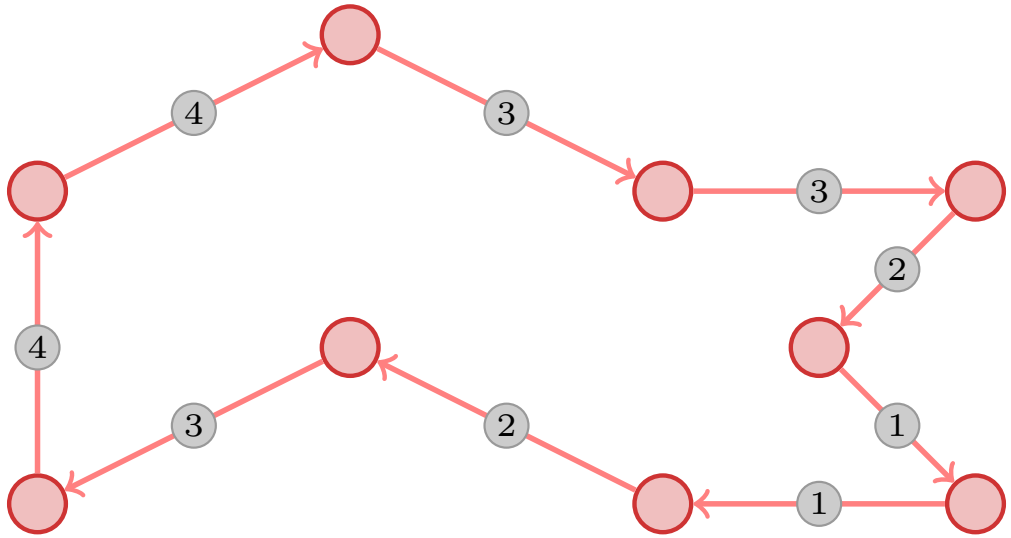
Voyageur de commerce (TSP)

TSP : Étant donné un ensemble de n villes et les distances qui les séparent, trouver la tournée la plus courte qui visite toutes les villes exactement une fois.

$\Leftrightarrow$ Trouver une permutation P de $\{0, \ldots, n-1\}$ qui minimise

$$\sum_{i=1}^{n-1} (d(P[i-1], P[i]) + d(P[n-1], P[0])$$

où $d(i, j)$ est égal à la distance entre la ville i et la ville j.



minimiser $\sum$ ●

Les listes

Variable de décision x ← list(n)

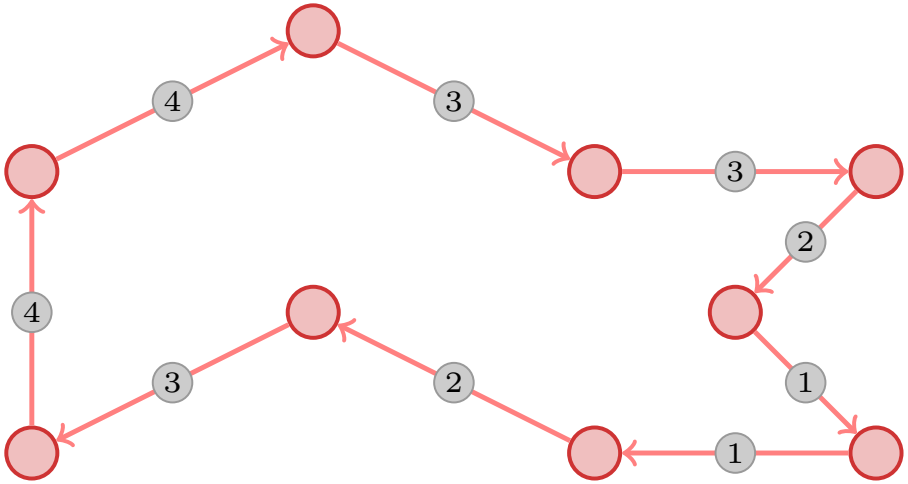
- Sous-ensemble ordonné du domaine $\{0, \ldots, n-1\}$
- Chaque valeur d'une liste est unique

Opérateurs sur les listes

- count(x) : nombre d'éléments dans la liste x
- x[index] : valeur à la position index dans la liste x (-1 si en dehors)

Voyageur de commerce (TSP)

```
function distance(city0, city1) {
  return sqrt(pow(x[city0]-x[city1],2), pow(y[city0]-y[city1],2));
}

function model() {
  route <- list(citiesCount);
  constraint count(route)==citiesCount;
  minimize sum[i in 1..citiesCount-1](distance(route[i], route[i-1]))
         + distance(route[citiesCount-1], route[0]);
}
```



minimiser $\sum$ ●

Voyageur de commerce (TSP)

```
function distance(city0, city1) {
  return sqrt(pow(x[city0]-x[city1],2), pow(y[city0]-y[city1],2));
}

function model() {
  route <- list(citiesCount);
  constraint count(route)==citiesCount;
  minimize sum[i in 1..citiesCount-1](distance(route[i], route[i-1]))
         + distance(route[citiesCount-1], route[0]);
}
```

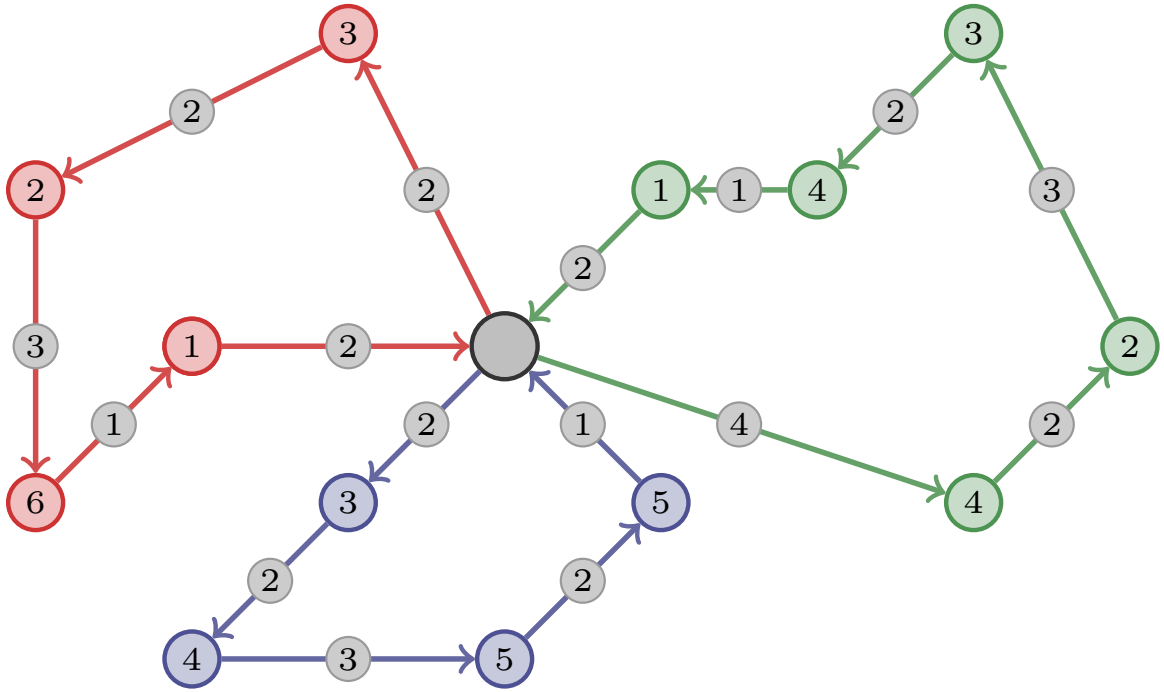
$\Leftrightarrow$ Trouver une permutation P de $\{0, \ldots, n-1\}$ qui minimise

$$\sum_{i=1}^{n-1} (d(P[i-1], P[i]) + d(P[n-1], P[0])$$

où $d(i, j)$ est égal à la distance entre la ville i et la ville j.

Tournées de véhicules avec contraintes de capacité (CVRP)

CVRP : Étant donné un ensemble de **clients**, les **distances** entre eux et des **demandes** à leur livrer, trouver un ensemble de **tournées** qui minimise la **somme des distances parcourues** et qui respecte la **capacité des camions**.

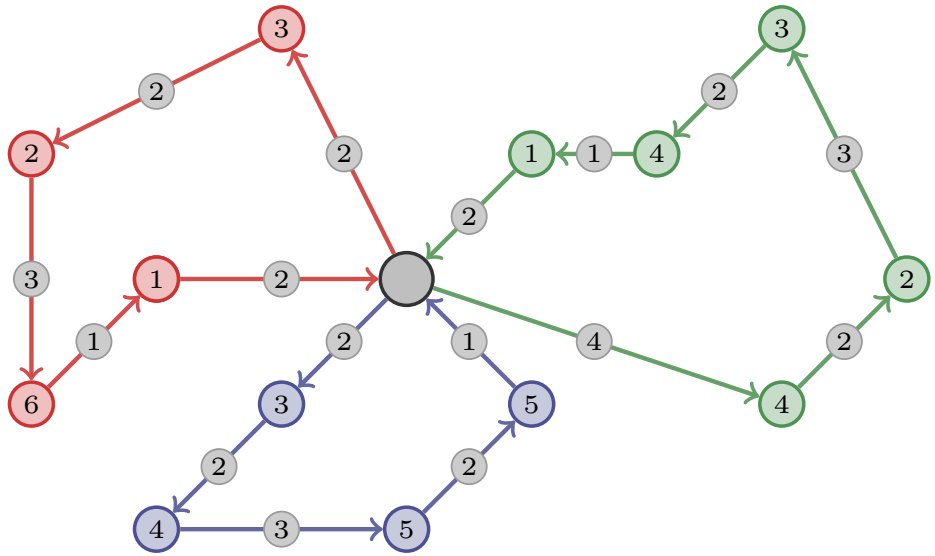


minimiser $\sum$ ● avec max($\sum$ ●, $\sum$ ●, $\sum$ ●) $\leq C$

Modélisation naturelle

Notions nécessaires :

- Ordonner des ensembles (permutations)
- Partitionner un ensemble
- Exprimer des valeurs à partir des sous-ensembles ordonnés de taille variable
- Contraindre ou optimiser ces valeurs



minimiser $\sum$ ● avec max($\sum$ ●, $\sum$ ●, $\sum$ ●) $\leq C$

Les listes

Variable de décision x $\leftarrow$ list(n)

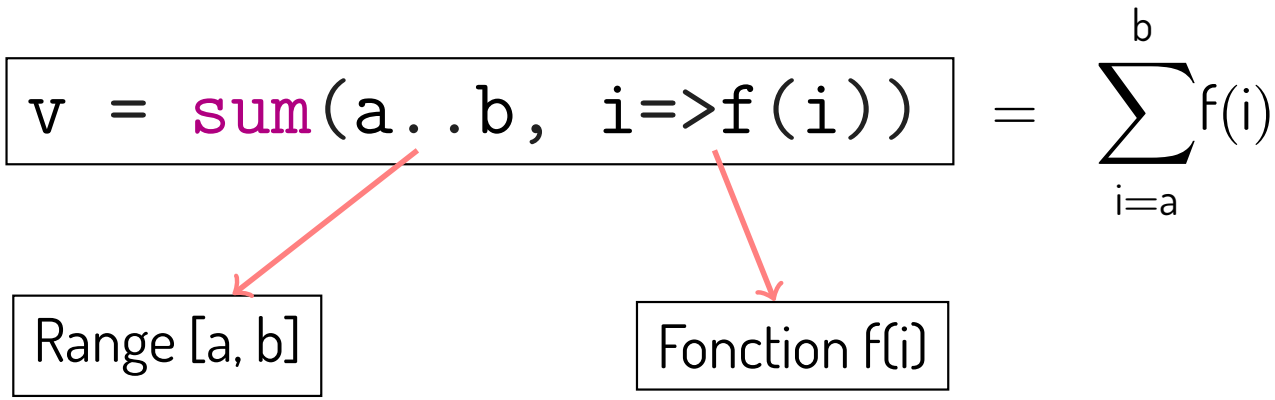
- Sous-ensemble ordonné du domaine $\{0, \ldots, n-1\}$
- Chaque valeur d'une liste est unique

Opérateurs sur les listes

- count(x) : nombre d'éléments dans la liste x
- x[index] : valeur à la position index dans la liste x (-1 si en dehors)
- partition($x_1, \ldots, x_k$) : vrai si les listes x_i partitionnent $\{0, \ldots, n-1\}$

Les opérateurs variadiques (LocalSolver 7.0)

Les opérateurs variadiques :

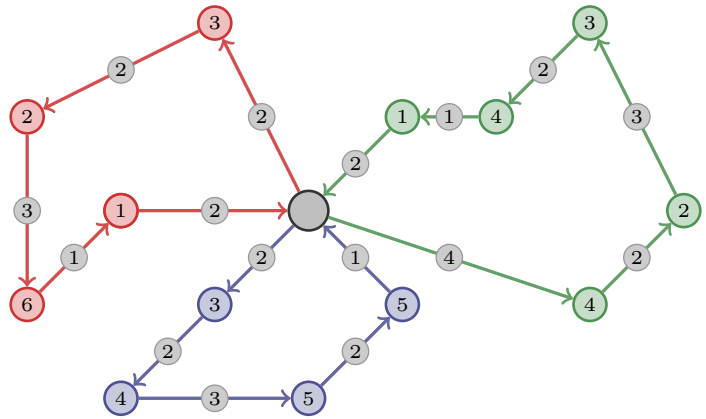


Exemple :

```
function model() {
  route <- list(clientsCount);
  demandsOnRoute = sum(0..count(route)-1, i => demands[route[i]]);
  /* ... */
```

Tournées de véhicules (CVRP)

```
function model() {
  routes[1..k] <- list(clientsCount);
  constraint partition[i in 1..k](routes[i]);
  for [r in 1..k] {
     route <- routes[r];
     l <- count(route);
     constraint sum(0..l-1, i => demands[route[i]]) <= capacity;
     dist[r] <- sum(0..l-2, i => distance(route[i], route[i+1]))
              + distance(depot, route[0]) + distance(route[l-1], depot);
  }
  minimize sum[r in 1..k](dist[r]);
}
```



minimiser $\sum$ ● avec max($\sum$ ●, $\sum$ ●, $\sum$ ●) $\leq C$

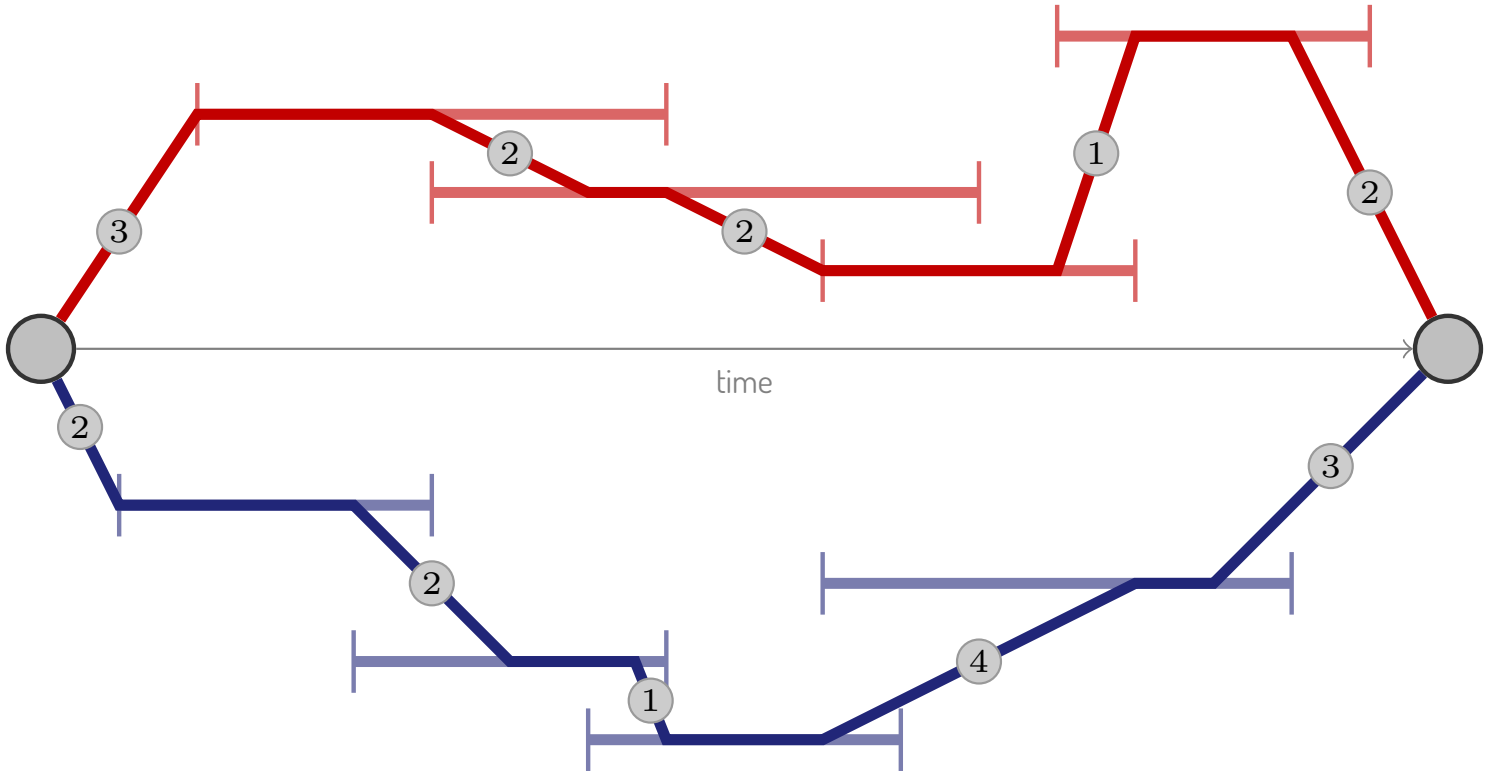
Tournées de véhicules avec fenêtres de temps (VRPTW)

VRPTW :

Étant donné :

- un ensemble de **clients** et les **distances** en temps qui les séparent,
- pour chaque client, une **fenêtre de temps** et **une durée de service**

trouver des tournées qui respectent les fenêtres de temps des clients.



Modélisation du problème

Calcul des dates d'arrivée et de départ des camions :

$$\text{arr}[\text{i}] = \begin{cases} \text{startTW}[\text{r}[\text{i}]], & \text{si i} = 0 \\ \max(\text{startTW}[\text{r}[\text{i}]], \text{dep}[\text{i}-1] + \text{distance}(\text{r}[\text{i}-1], \text{r}[\text{i}])), & \text{sinon} \end{cases}$$

$$\text{dep}[\text{i}] = \text{arr}[\text{i}] + \text{s}[\text{r}[\text{i}]]$$

où

- r est l'itinéraire d'un camion : un sous-ens. ordonné de $\{0, \ldots, \text{n}-1\}$
- startTW[c] est le début de la fenêtre de temps du client c
- s[c] est la durée de service pour le client c
- distance(c1, c2) est la durée nécessaire pour aller du client c1 au client c2

Les tableaux variadiques (LocalSolver 7.0)

Les tableaux variadiques définis récursivement :

```
v = array(a..b, (i, prev) => f(i, prev))
```

Range [a, b]

Fonction f(i, prev)

$$v[i] = f(i, v[i-1]) \text{ pour } i \in [a, b]$$

Tournées de véhicules avec fenêtres de temps (VRPTW)

```
function departureTime(route, i, prev) {
  arrivalTime <- (i==0) ? startTW[route[i]]
                        : max(startTW[route[i]],
                              prev + distance(route[i-1],route[i]));
  return arrivalTime + serviceTime[route[i]];
}
```

$$\text{arr}[i] = \begin{cases} \text{startTW}[r[i]], & \text{si } i = 0 \\ \max(\text{startTW}[r[i]], \text{dep}[i-1] + \text{distance}(r[i-1], r[i])), & \text{sinon} \end{cases}$$

$$\text{dep}[i] = \text{arr}[i] + s[r[i]]$$

Tournées de véhicules avec fenêtres de temps (VRPTW)

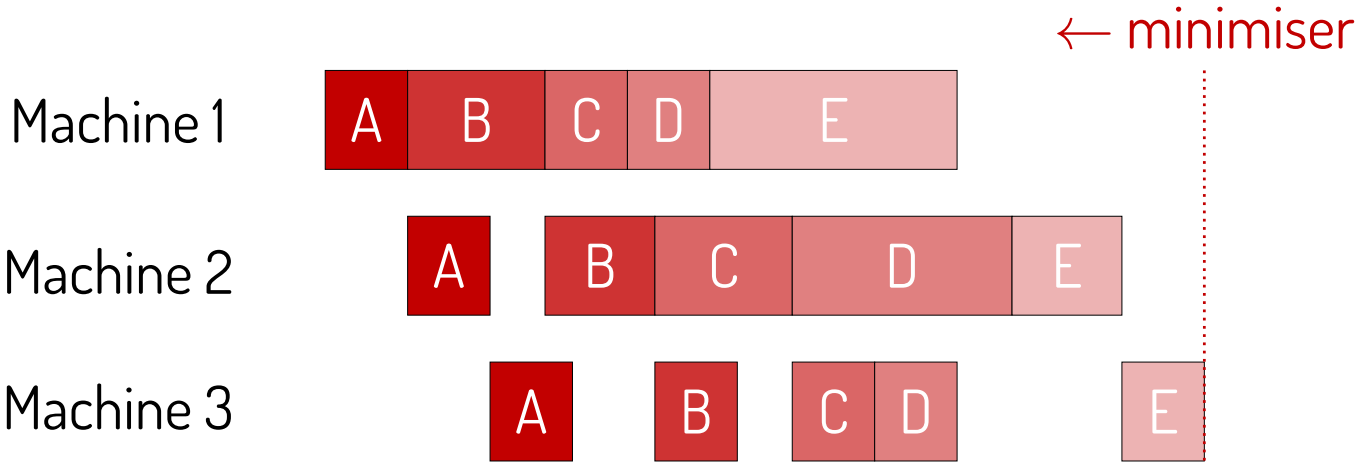
```
function departureTime(route, i, prev) {
  arrivalTime <- (i==0) ? startTW[route[i]]
                        : max(startTW[route[i]],
                              prev + distance(route[i-1],route[i]));
  return arrivalTime + serviceTime[route[i]];
}

function model() {
  routes[1..k] <- list(clientsCount);
  constraint partition[i in 1..k](routes[i]);
  for [r in 1..k] {
     route <- routes[r];
     l <- count(route);
     departureTimes <-
       array(0..l-1, (i, prev) => departureTime(route, i, prev));
     constraint and(0..l-1, i => departureTimes[i] <= endTW[route[i]]);
     /* ... */
```

Application à d'autres problèmes

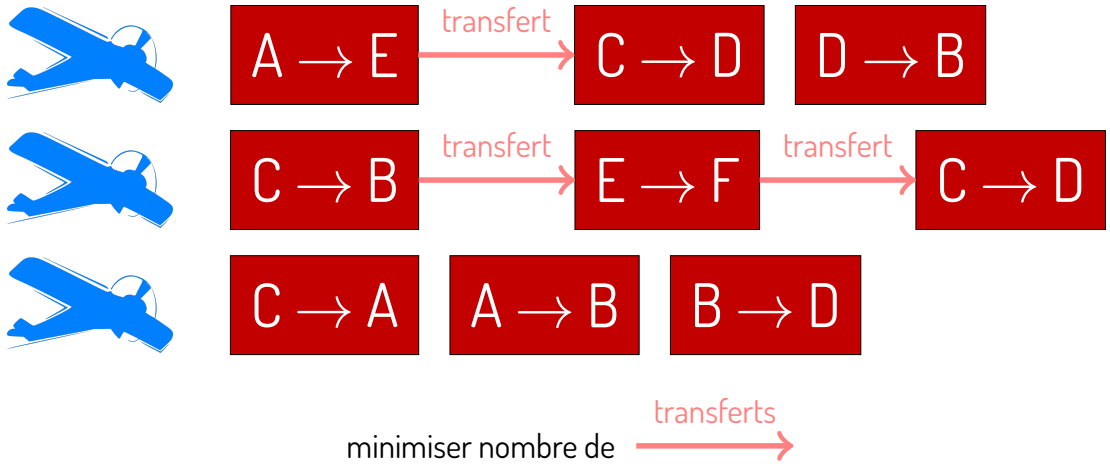
planification, ordonnancement...

Quelques problèmes

Flow-shop



Affectation des vols aux avions



Exemple Tour

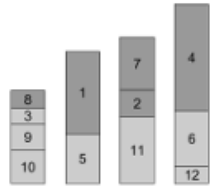
Smallest circle ★

Branin function ★

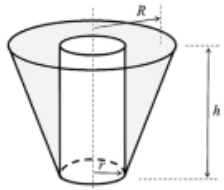
Max cut ★

Car sequencing ★★

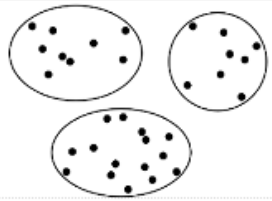
Social golfer ★★



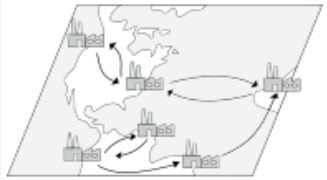
Steel mill slab design ★★



Optimal bucket ★★



K-means ★★



Quadratic assignment ★★

Travelling salesman ★★★

Flowshop ★★★

Vehicule routing ★★★

Previous

Next

www.localsolver.com

Conclusions

Décisions de type List :

- première étape vers la modélisation ensembliste

Opérateurs et tableaux variadiques :

- modélisation naturelle des problèmes
- Réduction de la mémoire et meilleures performances
- Meilleures compréhensions du modèle par le solveur

Perspectives :

- ensembles
- preprocessing, bornes inférieures
- détections de structures globales