

# Implémentation d'algorithmes de recherche opérationnelle. Projet phase 4

Polytechnique Montréal

MTH6412B

Xavier Lebeuf et Geoffroy Leconte

Novembre 2021

## Importation des fichiers

```
• begin
•     using Plots
•     using PlutoUI
•     using Test
•     using PrettyTables
•
•     include("mainphase4.jl")
• end;
• #hidden line
```

## 1. Structure AbstractSolution

Durant cette quatrième phase, lorsqu'un algorithme produit une solution, celle-ci peut souvent contenir beaucoup d'information. Nous avons donc choisis d'implémenter un nouveau type `AbstractSolution`. Ce surtype contient les deux types suivants.

Type contenant les informations d'une solution trouvée par l'algorithme RSL. Contient les attributs:

- cout::Float64 : coût minimal trouvé par l'algorithme,
- elapsed\_time::Float64 : temps de résolution,
- elapsed\_time\_cost::Float64 : temps de résolution incluant le calcul du coût en plus de la tournée,
- graphe::Graph{T, I} : graphe en sortie d'algorithme,

```

• begin
•     with_terminal() do
•         @doc RSLsolution
•     end
• end
• #hidden line

```

Type contenant les informations d'une solution trouvée par l'algorithme de Held et Karp. Contient les attributs:

- cout::Float64 : coût minimal trouvé par l'algorithme,
- status::String : status à la fin de l'algorithme,
- arbre::Vector{Union{Nothing, Edge{T, I}}}: 1-arbre en sortie d'algorithme (tour si status optimal),
- σw::Float64 : écart-type de w pour les wmemorysize dernières itérations,
- nbiter::Int : nombre d'itérations,
- elapsed\_time::Float64 : temps de résolution,
- wmemorysize::Int : paramètre pour la taille du vecteur d'écart-types des dernières itérations de la fonction,
- graphe::Graph{T, I} : graphe en sortie d'algorithme,

```

• begin
•     with_terminal() do
•         @doc Hksolution
•     end
• end
• #hidden line

```

## 2. Algorithme RSL

Notre fonction *rsl!* se base sur l'algorithme de Prim que nous avons amélioré pour cette phase en mémorisant les voisins directs des noeuds pour les graphes non complets.

```
rsl!(graph::Graph{T, I}, r::Node{T}) where {T, I}
```

Effectue l'algorithme de RSL en réordonnant les noeuds d'un graphe dans l'ordre de la tournée.

Renvoie une solution de type `RSLsolution{T, I} <: AbstractSolution{T, I}`.

```
• begin
•     with_terminal() do
•         @doc rsl!
•     end
• end
• #hidden line
```

Voici les résultats des tests unitaires avec la fonction `prim_acc!` (prim amélioré) sur l'exemple du laboratoire 3 ainsi que sur toutes les instances tsp symétriques. De plus, nous affichons le cout de l'arbre de cout minimal trouvé pour les instances tsp symétriques.

```
G exemple du cours ✓
bayg29.tsp✓ cout min: 1319.0
bays29.tsp✓ cout min: 1557.0
brazil58.tsp✓ cout min: 17514.0
brg180.tsp✓ cout min: 1920.0
dantzig42.tsp✓ cout min: 591.0
fri26.tsp✓ cout min: 741.0
gr120.tsp✓ cout min: 5805.0
gr17.tsp✓ cout min: 1421.0
gr21.tsp✓ cout min: 2161.0
gr24.tsp✓ cout min: 1011.0
gr48.tsp✓ cout min: 4082.0
hk48.tsp✓ cout min: 9905.0
pa561.tsp✓ cout min: 2396.0
swiss42.tsp✓ cout min: 1079.0
```

```
• begin
•     with_terminal() do
•         test_prim(Gexcours, a, prim_acc!) # test sur l'exemple du labo3
•         test_prim_all("instances/stsp", prim_acc!) # test tsp
•     end
• end
• #hidden line
```

Nous avons ensuite initialisé les enfants de chaque noeud grace aux parents de chaque noeud trouvés par l'algorithme de Prim, puis nous avons créé une fonction `parcours_preordre!` qui parcourt le graphe en pré-ordre de façon récursive. Elle ordonne les noeuds du graphe de manière à définir la tournée trouvée (l'ordre des noeuds donne l'ordre du parcours). Voici les résultats sur les problèmes de tsp symétriques. Pour chaque instance, l'algorithme est lancé sur le premier sommet du graphe ainsi que sur chaque sommet de l'arête de coût minimal. Pour chaque lancement, les tests unitaires sont vérifiés, le coût de la tournée minimale trouvée est affiché et le ratio avec la solution optimale est affiché.

```
bayg29.tsp
✓ cout minimal: 2.20e+03, 137% avec premier sommet
✓ cout minimal: 2.15e+03, 134% avec noeud1 arête cout min
✓ cout minimal: 2.15e+03, 134% avec noeud2 arête cout min
cout minimal: 1.61e+03 solution optimale
```

```
bays29.tsp
✓ cout minimal: 2.51e+03, 124% avec premier sommet
✓ cout minimal: 2.68e+03, 133% avec noeud1 arête cout min
✓ cout minimal: 2.68e+03, 133% avec noeud2 arête cout min
cout minimal: 2.02e+03 solution optimale
```

```
brazil58.tsp
✓ cout minimal: 3.03e+04, 119% avec premier sommet
✓ cout minimal: 2.94e+04, 116% avec noeud1 arête cout min
✓ cout minimal: 2.94e+04, 116% avec noeud2 arête cout min
cout minimal: 2.54e+04 solution optimale
```

brg180.tsp

```
• # test rsl!
• begin
•   with_terminal() do
•     test_rsl_all("instances/stsp", solutiontournee)
•   end
• end
```

Il est à noter que l'instance brg180.tsp a échoué les tests unitaires. Les ratios avec la solution optimale sont en effet bien supérieurs à 200%. Cela est une conséquence des coûts des arêtes qui ne respectent pas l'égalité triangulaire dans le graphe de cette instance. Observons ces trois sommets:

$u = \text{sommet } 2, v = \text{sommet } 3, w = \text{sommet } 1$

$$c(v, w) = 10^4, c(u, w) = 20, c(v, u) = 0$$

$$c(v, w) > c(v, u) + c(u, w)$$

## 3. Algorithme de Held et Karp (HK)

---

Pour implémenter cet algorithme, nous avons modifié notre type `Node` pour ajouter les poids  $\pi$  associés à chaque noeud, et le poids des voisins de chaque noeud.

A chaque itération l'algorithme transforme les noeuds du graphe pour ajouter les  $\pi_i$  actuels aux noeuds du graphe. Ensuite, nous enlevons un noeud et ses arêtes associées au graphe pour préparer la formation du 1-arbre. Ensuite, le code est séparé suivant l'utilisation de Prim ou de Kruskal. Dans les deux cas, nous construisons un arbre de coût minimal sur le graphe privé du noeud que nous avons enlevé plus haut, puis nous ajoutons ce noeud à l'arbre trouvé avec les 2 plus petites arêtes enlevées. Ensuite, nous calculons le coût de l'arbre et le coût  $\pi$  pour obtenir  $W$ , puis nous mettons à jour les degrés de chaque noeud et le vecteur  $\pi$ .

Les 3 critères d'arrêts choisis sont les suivants:

- Un certain nombre d'itération est atteint
- Tous les degrés des sommets du 1-arbre actuel sont de 2 (une tournée est trouvée)
- Les  $w$  des dernières itérations ne varient plus significativement. Ce dernier critère est vérifié par les paramètres `wmemorysize` et `σw`.

## 4. Paramètres de HK

---

```
hk!(graph::Graph{T, I}, r::Node{T};
    algorithm::Symbol=:prim, display::Bool=true,
    t0::Float64=50.0, maxiter::Int=nb_nodes(graph),
    wmemoriesize::Int=5, σw::Float64=1.0e-2) where {T, I}
```

Trouve une tournée de cout minimale par l'heuristique de Held et Karp. L'algorithme s'arrête lorsqu'une tournée maximale est trouvée, ou lorsque le nombre maximal d'itérations est atteint, ou lorsque l'écart-type des `wmemoriesize` dernières valeurs de `w` est plus faible que `σw`. L'algorithme renvoie la solution sous forme de `Hksolution{T, S} <: AbstractSolution{T, I}`.

- `graph::Graph{T, I}`: Graph complet dont on veut calculer la tournée minimale,
- `r::Node{T}`: noeud racine pour l'algorithme de Prim (ne doit pas être le dernier noeud), ignorer pour Kruskal,
- `algorithm::Symbol`: algorithme à utiliser pour le calcul de l'arbre de recouvrement minimal, choisir entre `:prim` (défaut) et `:kruskal`,
- `display::Bool`: active/désactive l'affichage (défaut `:true`),
- `t0::Float64`: valeur constante par laquelle la fonction décroissante `t1` est multipliée pour les mises à jour de  $\pi$ ,
- `maxiter::Int`: nombre maximal d'itérations,
- `wmemoriesize::Int`: paramètre pour la taille du vecteur d'écart-types des dernières itérations de la fonction,
- `σw::Float64`: tolérance pour l'écart-type des `wmemoriesize` dernières valeurs de `w`.

---

- `begin`
- `with_terminal() do`
- `@doc hk!`
- `end`
- `end`
- `#hidden line`

Voici la vérification des tests unitaires sur chaque instance tsp symétrique. On n'affiche pas le coût du 1-arbre trouvé ici car ce coût varie beaucoup avec les paramètres donnés. Les coûts avec des bons paramètres sont montrés dans une section ultérieure.

```

    ✓ avec Prim
gr21.tsp
    ✓ avec Kruskal
    ✓ avec Prim
gr24.tsp
    ✓ avec Kruskal
    ✓ avec Prim
gr48.tsp
    ✓ avec Kruskal
    ✓ avec Prim
hk48.tsp
    ✓ avec Kruskal
    ✓ avec Prim
pa561.tsp
    ✓ avec Kruskal
    ✓ avec Prim
swiss42.tsp
    ✓ avec Kruskal
    ✓ avec Prim

```

```

• begin
•     with_terminal() do
•         test_hk_all("instances/stsp", solutiontournee)
•     end
• end
• #hidden line

```

## Comparaisons entre Kruskal et Prim pour HK

Comparaison des performances entre les algorithmes implémentés

	temps prim	coût prim	temps kruskal	coût kruskal
bayg29	1.70e-02	1.39e+03	4.00e-03	1.39e+03
bays29	1.70e-02	1.87e+03	3.00e-03	1.82e+03
brazil58	1.93e-01	2.30e+04	3.90e-02	2.31e+04
brg180	5.31e+00	1.94e+03	3.24e+00	1.94e+03
dantzig42	7.10e-02	5.97e+02	1.40e-02	5.97e+02
fri26	2.00e-02	7.78e+02	3.00e-03	7.78e+02
gr120	1.62e+00	5.90e+03	4.69e-01	5.90e+03
gr17	4.00e-03	1.98e+03	1.00e-03	1.98e+03
gr21	7.00e-03	2.68e+03	1.00e-03	2.67e+03
gr24	9.00e-03	1.12e+03	1.00e-03	1.15e+03
gr48	8.30e-02	4.78e+03	1.30e-02	4.77e+03
hk48	7.20e-02	1.14e+04	1.20e-02	1.14e+04
pa561	2.92e+02	2.40e+03	4.71e+02	2.40e+03
swiss42	3.40e-02	1.11e+03	7.00e-03	1.11e+03

```

• begin
•     with_terminal() do
•         benchmark_table_KruskalPrim_hk(raw"./instances/stsp")
•     end
• end
• #hidden line

```

On constate que hk est plus rapide avec notre implémentation de Kruskal sur la plupart des problèmes TSP.

# 6. Meilleurs paramètres et illustrations graphiques

---

Voici maintenant les meilleurs paramètres pour chaque algorithme sur chaque instance avec une représentation graphique si des coordonées sont données pour les sommets. Le  $\Delta$  relatif représente l'erreur relative par rapport à une tournée optimale. Ces solutions optimales sont tirées du fichier `solutions_stsp.txt`.

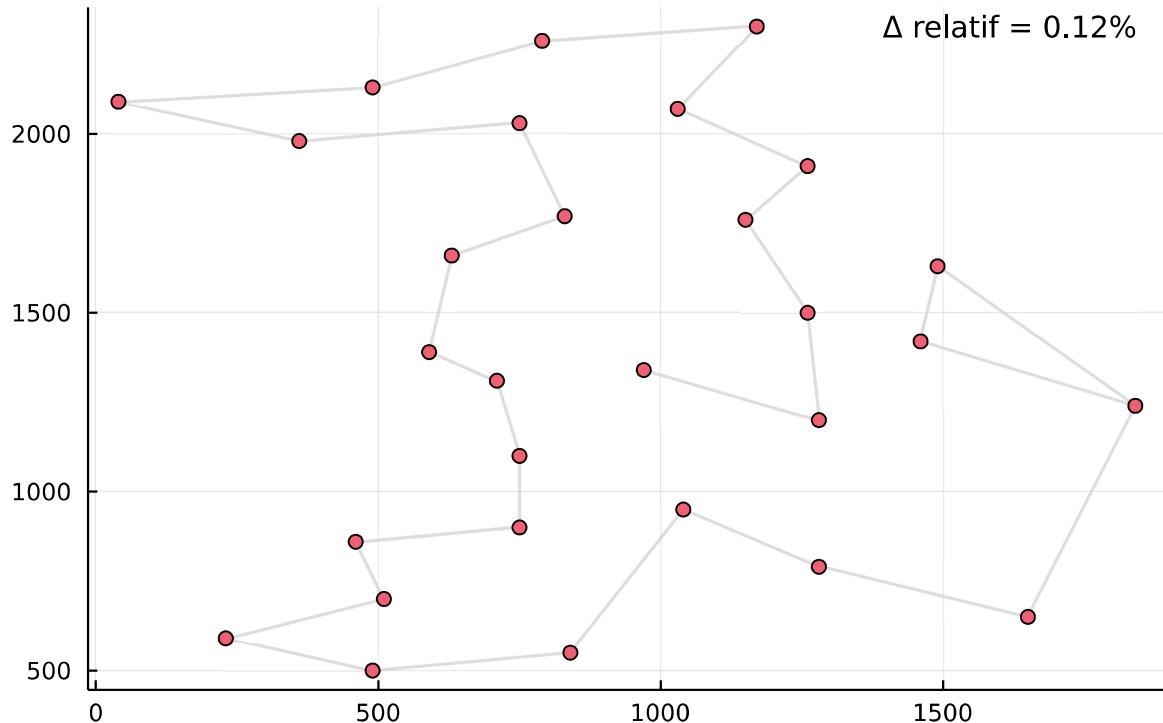
L'algorithme de RSL ne calcule pas le coût de la tournée trouvée, et le temps de ce calcul est en  $O(|A||S|)$ . En affichant une solution trouvée par l'algorithme, nous montrons donc le "Temps sans cout" qui est le temps pris par l'algorithme seulement, et le "Temps total" qui inclut le calcul du coût total.

## bayg29

---

### Held et Karp

# 1-arbre dans bayg29



```

• begin
•     solution1 = hk("bayg29", raw"instances/stsp/bayg29.tsp",
•                     racine=:premier,
•                     algorithm=:kruskal,
•                     display=false,
•                     t0=50.0,
•                     maxiter=300,
•                     wmemorysize=5,
•                     σw=1.0e-3)
•     plot_tour_gap(solution1, solutiontournee)
• end

```

Cout = 1607.993896484375  
 Status = non increasing values in last iterations  
 $\sigma$  des 5 dernières itérations = 0.0009826470604247122  
 Nombre d'itérations = 200  
 Temps total (s) = 0.015000104904174805

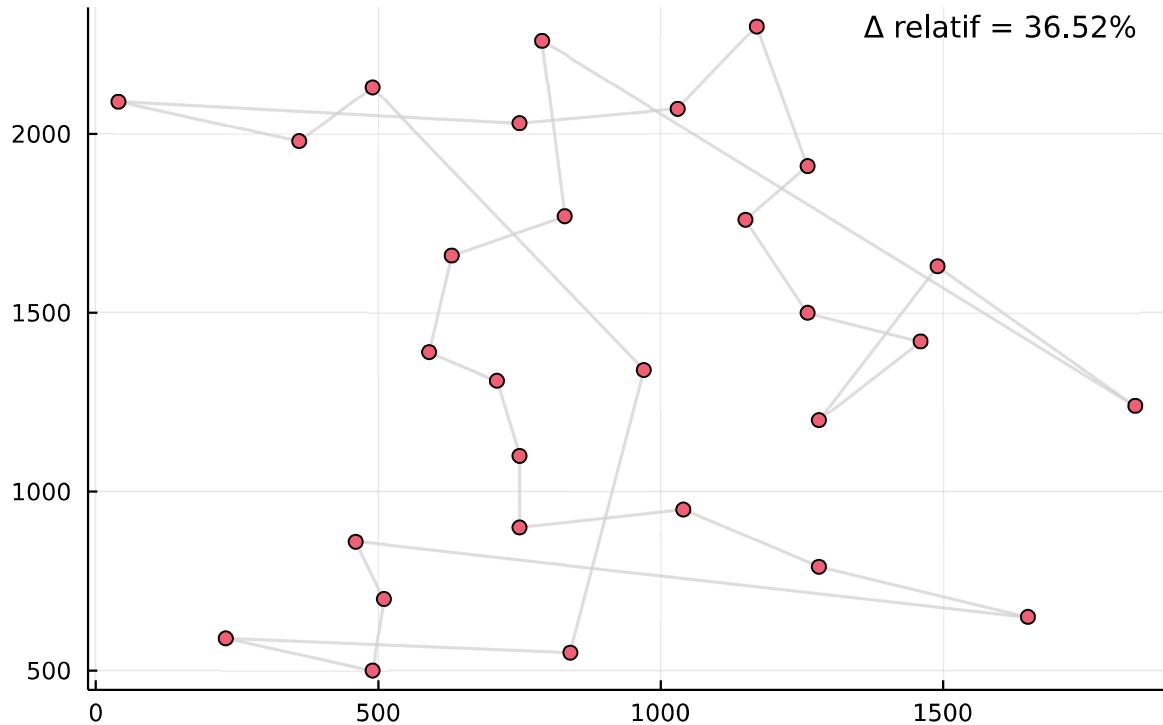
```

• begin
•     with_terminal() do
•         println(solution1)
•     end
• end

```

## Rosenkrantz, Stearns et Lewis

## tournée dans bayg29



```

• begin
•     solution2 = rsl("bayg29", raw"instances/stsp/bayg29.tsp", racine=:premier)
•     plot_tour_gap(solution2, solutiontournee)
• end

```

```

Cout = 2198.0
Temps sans cout (s) = 0.0009999275207519531
Temps total (s) = 0.0009999275207519531

```

```

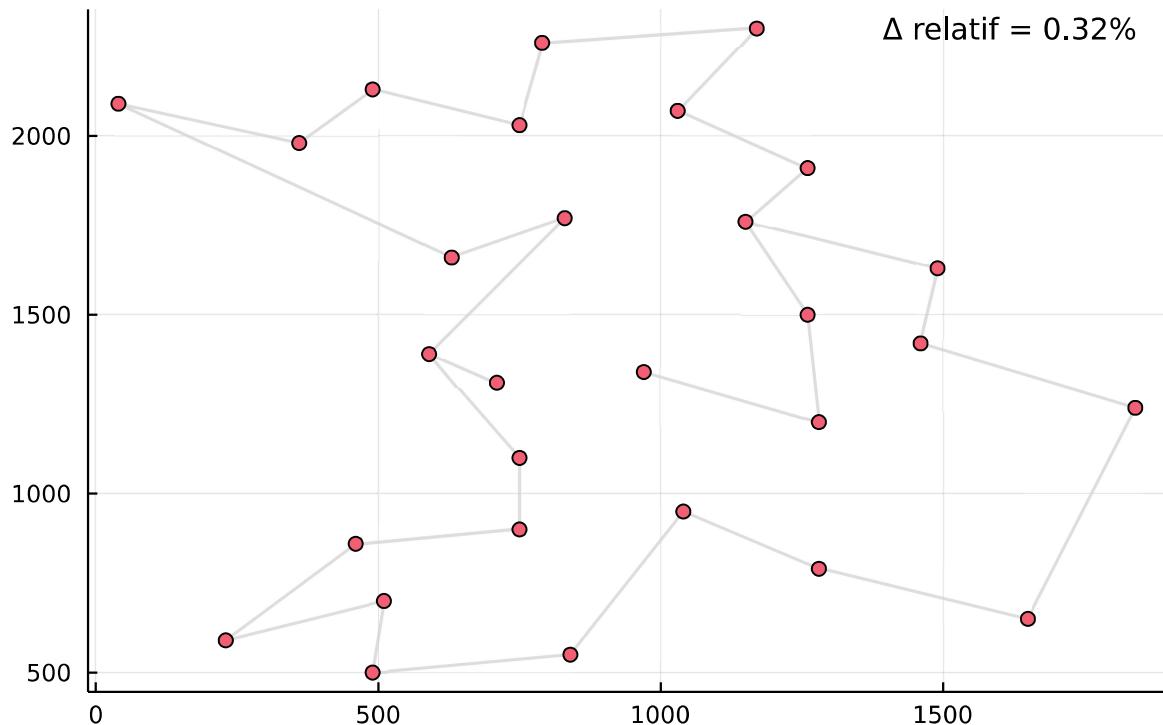
• begin
•     with_terminal() do
•         println(solution2)
•     end
• end

```

## bays29

Held et Karp

# 1-arbre dans bays29



```

• begin
•     solution3 = hk("bays29", raw"instances/stsp/bays29.tsp",
•                     racine=:premier,
•                     algorithm=:prim,
•                     display=false,
•                     t0=75.0,
•                     maxiter=300,
•                     wmemorysize=5,
•                     σw=1.0e-4)
•     plot_tour_gap(solution3, solutiontournee)
• end

```

Cout = 2013.4989166259766  
 Status = non increasing values in last iterations  
 $\sigma$  des 5 dernières itérations = 6.686066375795486e-5  
 Nombre d'itérations = 261  
 Temps total (s) = 0.1510000228881836

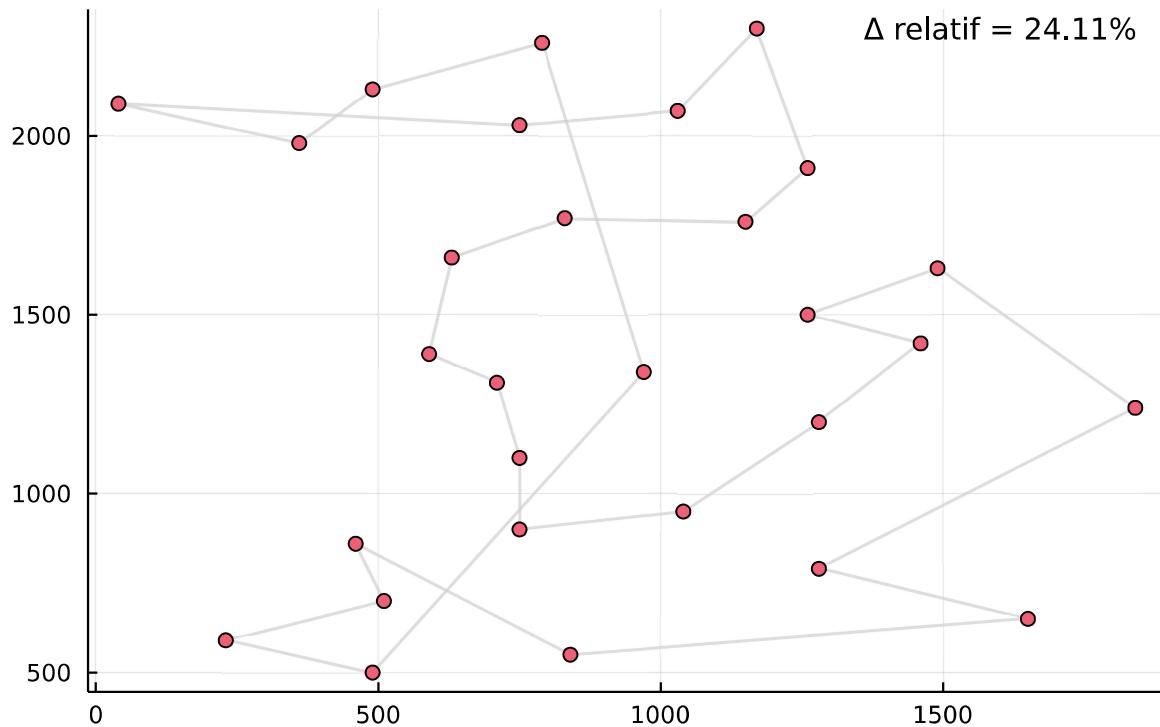
```

• begin
•     with_terminal() do
•         println(solution3)
•     end
• end

```

## Rosenkrantz, Stearns et Lewis

## tournée dans bays29



```

• begin
•     solution4 = rsl("bays29", raw"instances/stsp/bays29.tsp", racine=:premier)
•     plot_tour_gap(solution4, solutiontournee)
• end

```

Cout = 2507.0  
Temps sans cout (s) = 0.0  
Temps total (s) = 0.0

```

• begin
•     with_terminal() do
•         println(solution4)
•     end
• end

```

## brazil58

Held et Karp

```
Cout = 25354.38427734375
Status = non increasing values in last iterations
σ des 5 dernières itérations = 0.006953509030827304
Nombre d'itérations = 383
Temps total (s) = 0.14700007438659668
Δrelatif = 0.1599359033520378
```

```
• begin
•     solution5 = hk("brazil58", raw"instances/stsp/brazil58.tsp",
•                     racine=:premier,
•                     algorithm=:kruskal,
•                     display=false,
•                     t0=500.0,
•                     maxiter=500,
•                     wmemoriesize=5,
•                     σw=1.0e-2)
•     with_terminal() do
•         println(solution5)
•         Δcout(solution5, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 30336.0
Temps sans cout (s) = 0.003000020980834961
Temps total (s) = 0.003000020980834961
Δrelatif = 19.45658594211459
```

```
• begin
•     solution6 = rsl("brazil58", raw"instances/stsp/brazil58.tsp", racine=:premier)
•     with_terminal() do
•         println(solution6)
•         Δcout(solution6, solutiontournee)
•     end
• end
```

## brg180

### Held et Karp

```
Cout = 1940.0
Status = non increasing values in last iterations
σ des 5 dernières itérations = 0.8366600265340756
Nombre d'itérations = 44
Temps total (s) = 0.40400004386901855
Δrelatif = 0.5128205128205128
```

```
• begin
•     solution7 = hk("brg180", raw"instances/stsp/brg180.tsp",
•                     racine=:premier,
•                     algorithm=:kruskal,
•                     display=false,
•                     t0=1.0,
•                     maxiter=300,
•                     wmemoriesize=5,
•                     σw=1.0)
•     with_terminal() do
•         println(solution7)
•         Δcout(solution7, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

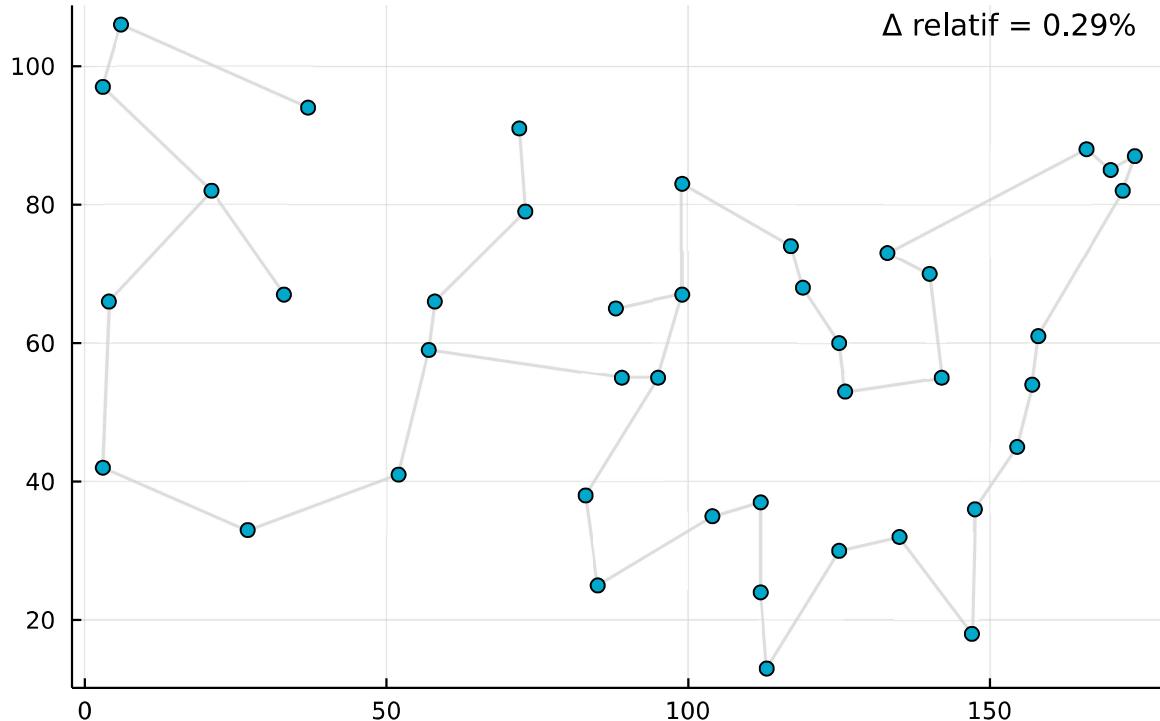
```
Cout = 116860.0
Temps sans cout (s) = 0.016000032424926758
Temps total (s) = 0.01900005340576172
Δrelatif = 5892.820512820513
```

```
• begin
•     solution8 = rsl("brg180", raw"instances/stsp/brg180.tsp", racine=:premier)
•     with_terminal() do
•         println(solution8)
•         Δcout(solution8, solutiontournee)
•     end
• end
```

## dantzig42

### Held et Karp

# 1-arbre dans dantzig42



```

• begin
•     solution9 = hk("dantzig42", raw"instances/stsp/dantzig42.tsp",
•                     racine=:premier,
•                     algorithm=:kruskal,
•                     display=false,
•                     t0=30.0,
•                     maxiter=300,
•                     wmemorysize=5,
•                     σw=1.0e-3)
•     plot_tour_gap(solution9, solutiontournee)
• end

```

```

Cout = 696.99365234375
Status = max iteration
σ des 5 dernières itérations = 0.0012579758816394775
Nombre d'itérations = 300
Temps total (s) = 0.05300021171569824

```

```

• begin
•     with_terminal() do
•         println(solution9)
•     end
• end

```

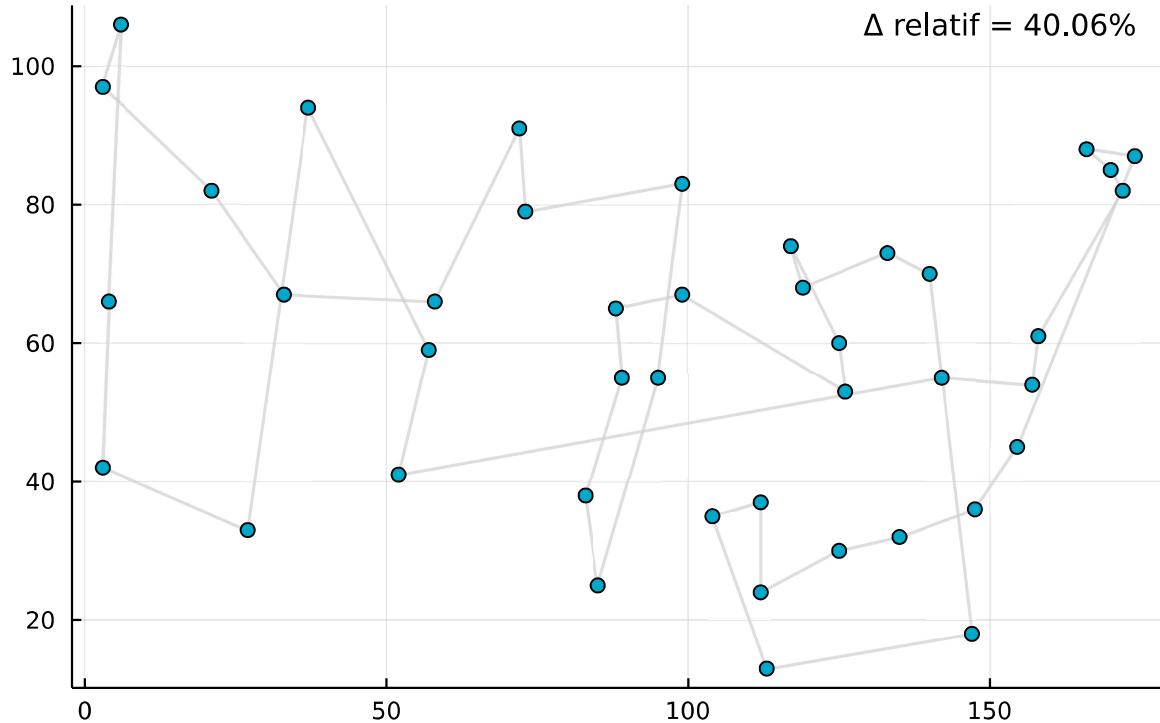
## Rosenkrantz, Stearns et Lewis

```

• md"""
• #### Rosenkrantz, Stearns et Lewis
• """

```

## tournée dans dantzig42



```

• begin
•     solution10 = rsl("dantzig42", raw"instances/stsp/dantzig42.tsp",
•     racine=:premier)
•     plot_tour_gap(solution10, solutiontournee)
end

```

Cout = 979.0  
Temps sans cout (s) = 0.0009999275207519531  
Temps total (s) = 0.0009999275207519531

## fri26

### Held et Karp

```
Cout = 936.9999694824219
Status = max iteration
σ des 5 dernières itérations = 1.88045616819248e-5
Nombre d'itérations = 300
Temps total (s) = 0.01399993896484375
Δrelatif = 3.2569453708644612e-6
```

```
• begin
•     solution11 = hk("fri26", raw"instances/stsp/fri26.tsp",
•                         racine=5,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=200.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-5)
•     with_terminal() do
•         println(solution11)
•         Δcout(solution11, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 1141.0
Temps sans cout (s) = 0.0
Temps total (s) = 0.0
Δrelatif = 21.77161152614728
```

```
• begin
•     solution12 = rsl("fri26", raw"instances/stsp/fri26.tsp", racine=:premier)
•     with_terminal() do
•         println(solution12)
•         Δcout(solution12, solutiontournee)
•     end
• end
```

## gr17

### Held et Karp

```
Cout = 2085.0
Status = optimal
σ des 5 dernières itérations = 0.46192954684670257
Nombre d'itérations = 67
Temps total (s) = 0.0009999275207519531
Δrelatif = 0.0
```

```
• begin
•     solution13 = hk("gr17", raw"instances/stsp/gr17.tsp",
•                         racine=5,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=30.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-3)
•     with_terminal() do
•         println(solution13)
•         Δcout(solution13, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 2352.0
Temps sans cout (s) = 0.0
Temps total (s) = 0.0
Δrelatif = 12.805755395683452
```

```
• begin
•     solution14 = rsl("gr17", raw"instances/stsp/gr17.tsp", racine=:premier)
•     with_terminal() do
•         println(solution14)
•         Δcout(solution14, solutiontournee)
•     end
• end
```

## gr21

### Held et Karp

```
Cout = 2707.0
Status = optimal
σ des 5 dernières itérations = 1.6312380267759823
Nombre d'itérations = 48
Temps total (s) = 0.0010001659393310547
Δrelatif = 0.0
```

```
• begin
•     solution15 = hk("gr21", raw"instances/stsp/gr21.tsp",
•                         racine=5,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=30.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-3)
•     with_terminal() do
•         println(solution15)
•         Δcout(solution15, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 3728.0
Temps sans cout (s) = 0.0
Temps total (s) = 0.0
Δrelatif = 37.71702992242334
```

```
• begin
•     solution16 = rsl("gr21", raw"instances/stsp/gr21.tsp", racine=:premier)
•     with_terminal() do
•         println(solution16)
•         Δcout(solution16, solutiontournee)
•     end
• end
```

## gr24

### Held et Karp

```
Cout = 1271.990966796875
Status = non increasing values in last iterations
σ des 5 dernières itérations = 0.0009704397907398336
Nombre d'itérations = 163
Temps total (s) = 0.009000062942504883
Δrelatif = 0.0007101574783805032
```

```
• begin
•     solution17 = hk("gr24", raw"instances/stsp/gr24.tsp",
•                         racine=5,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=30.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-3)
•     with_terminal() do
•         println(solution17)
•         Δcout(solution17, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 1641.0
Temps sans cout (s) = 0.0010001659393310547
Temps total (s) = 0.0010001659393310547
Δrelatif = 29.009433962264154
```

```
• begin
•     solution18 = rsl("gr24", raw"instances/stsp/gr24.tsp", racine=:premier)
•     with_terminal() do
•         println(solution18)
•         Δcout(solution18, solutiontournee)
•     end
• end
```

## gr48

### Held et Karp

```
Cout = 4958.984375
Status = max iteration
σ des 5 dernières itérations = 0.004279082480509111
Nombre d'itérations = 300
Temps total (s) = 0.06500005722045898
Δrelatif = 1.724447582243361
```

```
• begin
•     solution19 = hk("gr48", raw"instances/stsp/gr48.tsp",
•                         racine=:premier,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=20.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-3)
•     with_terminal() do
•         println(solution19)
•         Δcout(solution19, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

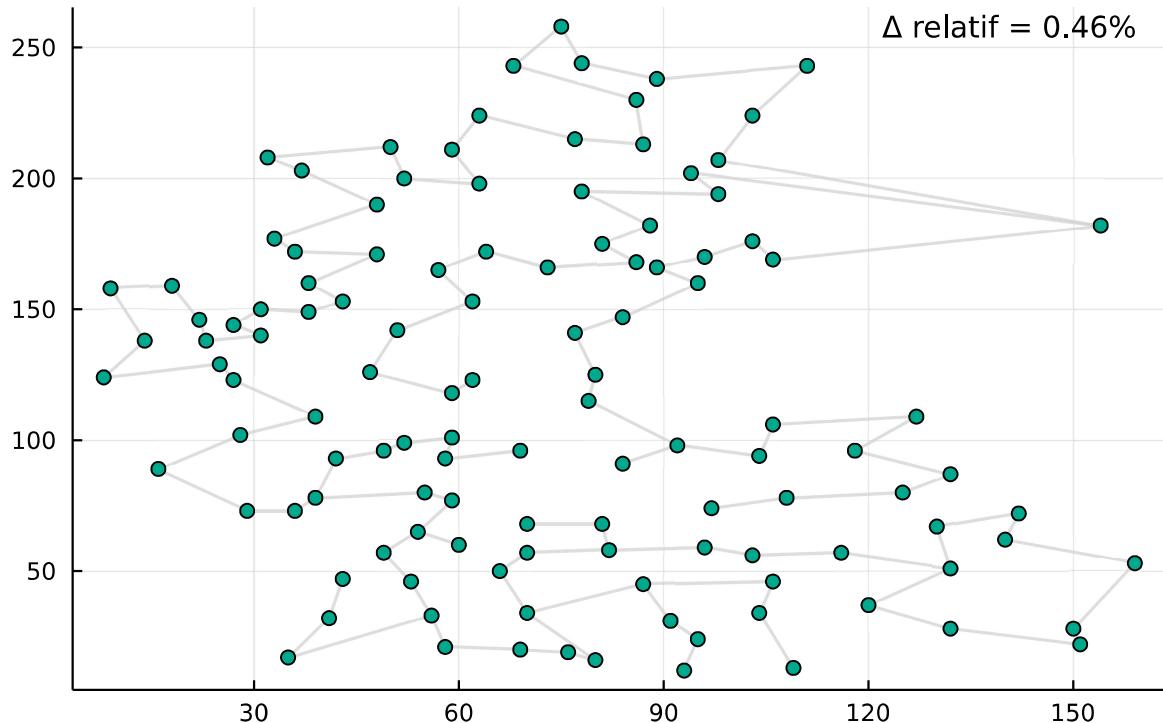
```
Cout = 6898.0
Temps sans cout (s) = 0.0019998550415039062
Temps total (s) = 0.0019998550415039062
Δrelatif = 36.702338485929445
```

```
• begin
•     solution20 = rsl("gr48", raw"instances/stsp/gr48.tsp", racine=:premier)
•     with_terminal() do
•         println(solution20)
•         Δcout(solution20, solutiontournee)
•     end
• end
```

## gr120

### Held et Karp

# 1-arbre dans gr120



```

• begin
•     solution21 = hk("gr120", raw"instances/stsp/gr120.tsp",
•                         racine=10,
•                         algorithm=:prim,
•                         display=false,
•                         t0=5.0,
•                         maxiter=500,
•                         wmemorysize=5,
•                         σw=1.0e-2)
•     plot_tour_gap(solution21, solutiontournee)
• end

```

```

Cout = 6910.0078125
Status = max iteration
σ des 5 dernières itérations = 0.047312412276999255
Nombre d'itérations = 500
Temps total (s) = 4.922000169754028

```

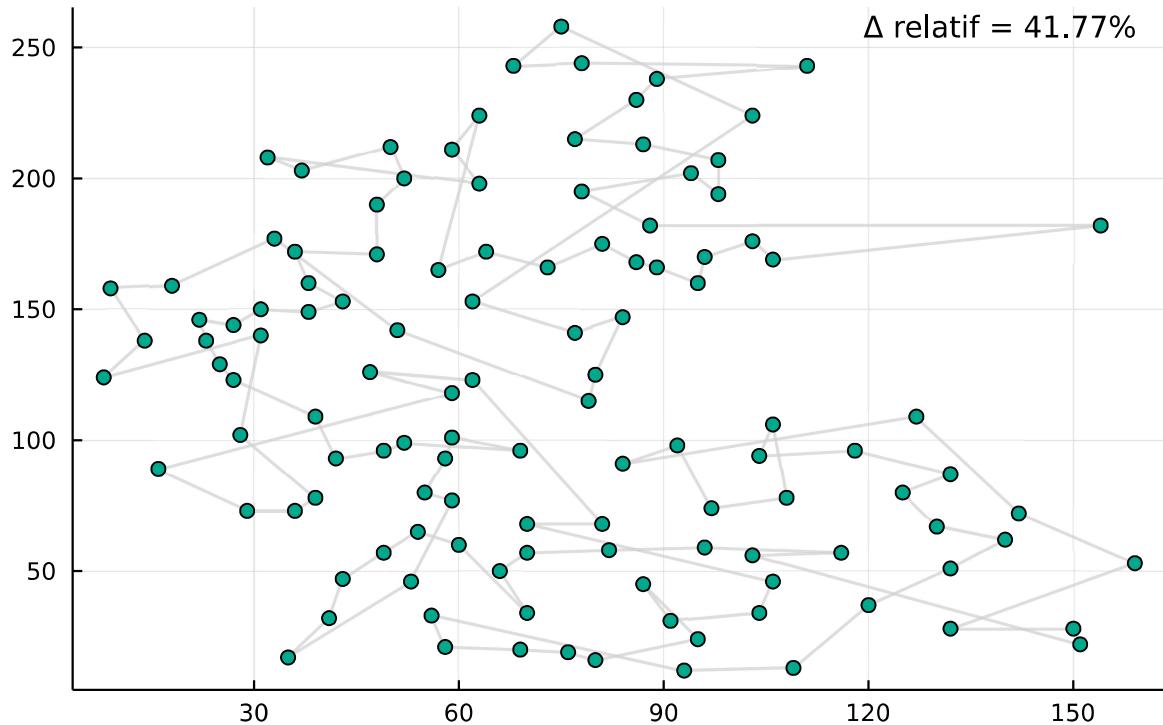
```

• begin
•     with_terminal() do
•         println(solution21)
•     end
• end

```

## Rosenkrantz, Stearns et Lewis

## tournée dans gr120



```

• begin
•     solution22 = rsl("gr120", raw"instances/stsp/gr120.tsp", racine=:premier)
•     plot_tour_gap(solution22, solutiontournee)
• end

```

Cout = 9842.0  
Temps sans cout (s) = 0.006999969482421875  
Temps total (s) = 0.00800013542175293

```

• begin
•     with_terminal() do
•         println(solution22)
•     end
• end

```

## hk48

### Held et Karp

```
Cout = 11444.359375
Status = non increasing values in last iterations
σ des 5 dernières itérations = 0.008558164961018222
Nombre d'itérations = 210
Temps total (s) = 0.04500007629394531
Δrelatif = 0.14519348224413228
```

```
• begin
•     solution23 = hk("hk48", raw"instances/stsp/hk48.tsp",
•                         racine=10,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=20.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-2)
•     with_terminal() do
•         println(solution23)
•         Δcout(solution23, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

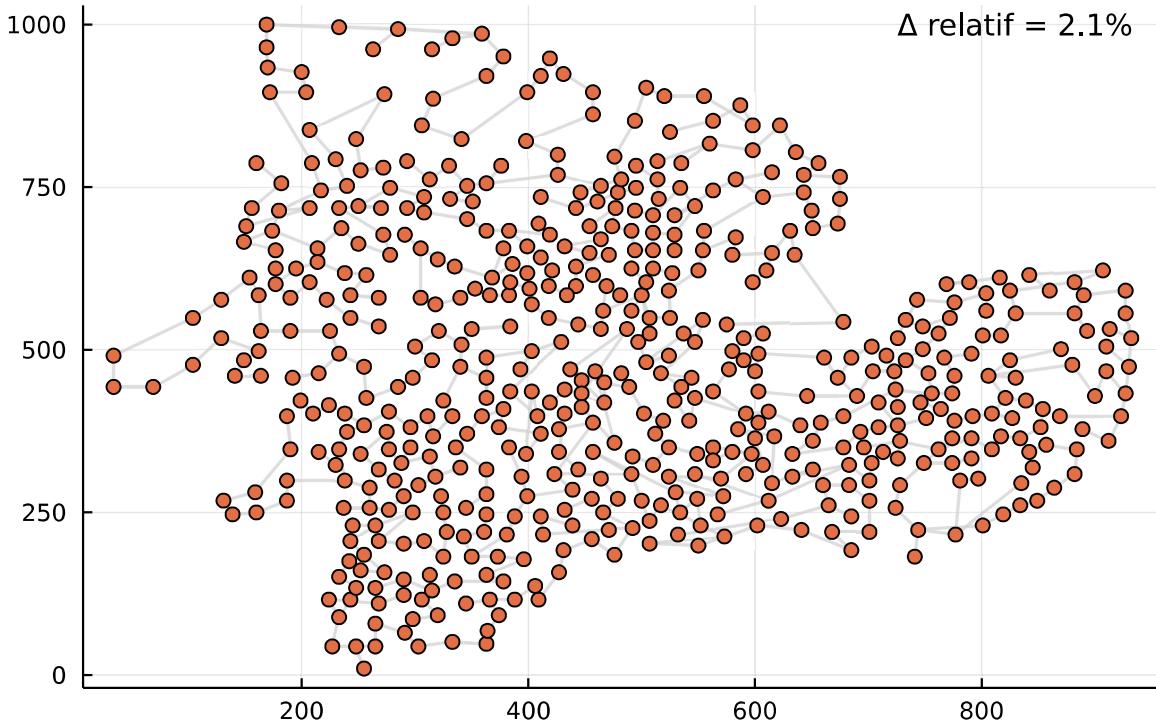
```
Cout = 14905.0
Temps sans cout (s) = 0.0010001659393310547
Temps total (s) = 0.0010001659393310547
Δrelatif = 30.049733880115177
```

```
• begin
•     solution24 = rsl("hk48", raw"instances/stsp/hk48.tsp", racine=:premier)
•     with_terminal() do
•         println(solution24)
•         Δcout(solution24, solutiontournee)
•     end
• end
```

## pa561

### Held et Karp

# 1-arbre dans pa561



```

• begin
•     solution25 = hk("pa561", raw"instances/stsp/pa561.tsp",
•                         racine=200,
•                         algorithm=:prim,
•                         display=false,
•                         t0=1.0,
•                         maxiter=300,
•                         wmemorysize=5,
•                         σw=1.0e-2)
•     plot_tour_gap(solution25, solutiontournee)
• end

```

Cout = 2705.0  
 Status = max iteration  
 $\sigma$  des 5 dernières itérations = 1.9170289512680814  
 Nombre d'itérations = 300  
 Temps total (s) = 173.10599994659424

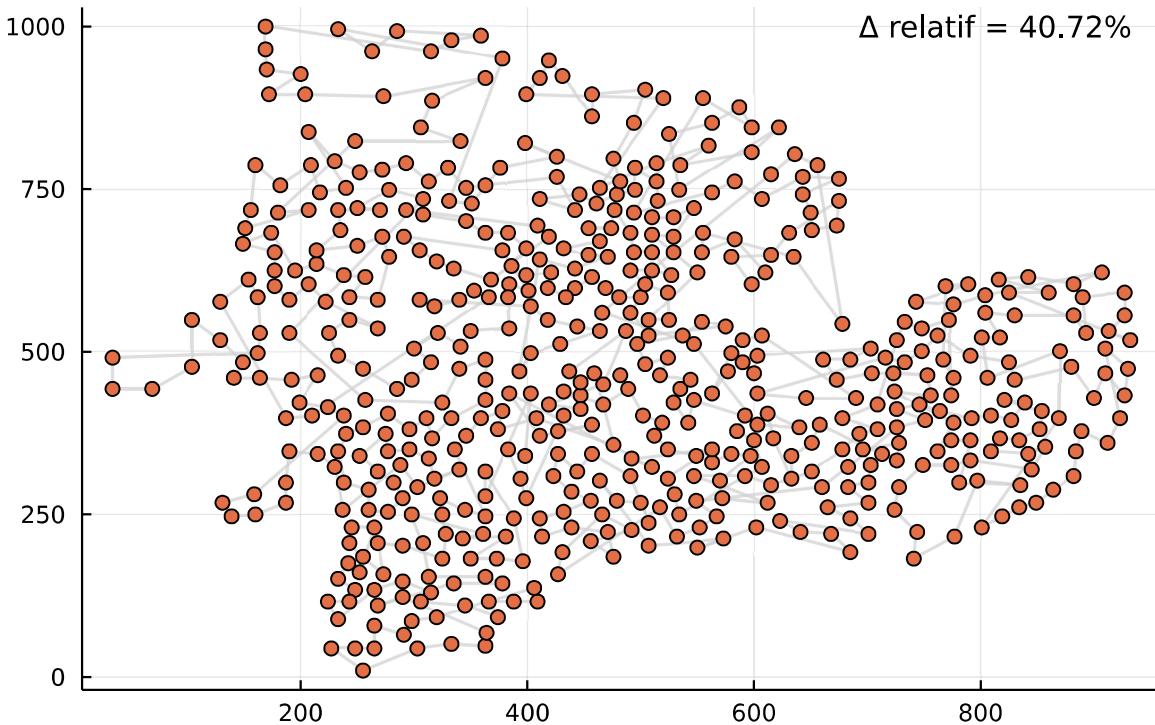
```

• begin
•     with_terminal() do
•         println(solution25)
•     end
• end

```

## Rosenkrantz, Stearns et Lewis

## tournée dans pa561



```

• begin
•     solution26 = rsl("pa561", raw"instances/stsp/pa561.tsp", racine=:premier)
•     plot_tour_gap(solution26, solutiontournee)
• end

```

Cout = 3888.0  
Temps sans cout (s) = 0.19700002670288086  
Temps total (s) = 0.2839999198913574

```

• begin
•     with_terminal() do
•         println(solution26)
•     end
• end

```

## swiss42

Held et Karp

```
Cout = 1271.9994506835938
Status = non increasing values in last iterations
σ des 5 dernières itérations = 0.000938235630066643
Nombre d'itérations = 298
Temps total (s) = 0.04799985885620117
Δrelatif = 0.0785977467718971
```

```
• begin
•     solution27 = hk("swiss42", raw"instances/stsp/swiss42.tsp",
•                         racine=:premier,
•                         algorithm=:kruskal,
•                         display=false,
•                         t0=5.0,
•                         maxiter=300,
•                         wmemoriesize=5,
•                         σw=1.0e-3)
•     with_terminal() do
•         println(solution27)
•         Δcout(solution27, solutiontournee)
•     end
• end
```

## Rosenkrantz, Stearns et Lewis

```
Cout = 1685.0
Temps sans cout (s) = 0.0009999275207519531
Temps total (s) = 0.0009999275207519531
Δrelatif = 32.36449332285939
```

```
• begin
•     solution28 = rsl("swiss42", raw"instances/stsp/swiss42.tsp", racine=:premier)
•     with_terminal() do
•         println(solution28)
•         Δcout(solution28, solutiontournee)
•     end
• end
```

# 7. Reproduction des résultats

Tel que démontré dans la section précédente, il suffit d'appeler `rsl()` ou `hk()` avec le nom que l'on désire donner au graphe, le chemin vers l'instance que l'on veut tester, puis les paramètres voulus pour obtenir une solution. Il est possible de n'entrer aucun paramètre pour laisser ceux par défaut.

L'appel de la fonction `plot_tour_gap` illustre cette solution graphiquement.

```
hk(graphname::String, path::String; racine::Symbol=:premier, kwargs...)
```

Fonction faisant appel à `hk!` en prenant en argument un chemin `path` vers le fichier TSP à résoudre, et un nom de graphe `graphname`. L'argument `racine::Symbol` doit être un `Int`, `:premier`, `:dernier` (fonctionne seulement pour Kruskal), ou `:poidsmin` (noeud choisi sur l'arête de coût minimal). `kwargs...` permet de spécifier les arguments optionnels de `hk!` comme `t0`, etc

...

```
• begin
•     with_terminal() do
•         @doc hk
•     end
• end
• #hidden line
```

```
rsl(graphname::String, path::String; racine::Symbol=:premier)
```

Fonction faisant appel à `rsl!` en prenant en argument un chemin `path` vers le fichier TSP à résoudre, et un nom de graphe `graphname`. L'argument `racine` doit être un `Int`, `:premier`, `:dernier` ou `:poidsmin` (noeud choisi sur l'arête de coût minimal).

```
• begin
•     with_terminal() do
•         @doc rsl
•     end
• end
• #hidden line
```

## 8. Branche phase4 sur GitHub

Lien: <https://github.com/XavierLebeuf/mth6412b-starter-code/tree/phase4>

