MTH6412B: Projet Voyageur de Commerce

Phase 4: Tournées minimales

- **Auteur**: El Hadji Abdou Aziz Ndiaye (1879468)
- **Code source**: **Repertoire Github**

Importation du code

```
run_test_tsp_hk (generic function with 1 method)
```

```
begin
    using PlutoUI
    using Plots
    using Printf
    include("node.jl")
    include("edge.jl")
    include("read_stsp.jl")
    include("graph.jl")
    include("mst_kruskal.jl")
    include("mst_prim.jl")
    include("tsp_rsl.jl")
    include("tsp_hk.jl")
    include("tsp.jl")
    include("tests/test_node.jl")
    include("tests/test_edge.jl")
    include("tests/test_graph.jl")
    include("tests/test_mst_kruskal.jl")
    include("tests/test_mst_prim.jl")
    include("tests/test_tsp_rsl.jl")
    include("tests/test_tsp_hk.jl")
```

Révision du code de la phase 3

Le code de la phase 3 n'a pas beaucoup été modifié.

L'algorithme de calcul des listes d'adjacence des arêtes à été légerement améliorer pour gérer plus facilement les 1-trees dans l'algorithme de Held et Karp (HK).

Un mutateur de l'attribut `weight` de la structure `Edge` à été ajouté pour tenir compte des nouveaux algorithmes de tournées minimales.

Les tests des différentes structures modifiées ont été mis à jour.

Tests des structures de base (**`Node`**, **`Edge`** et **`Graph`**):

```
Test de la structure de données `Node` : -v
Test de la structure de données `Edge` : -v
Test de la structure de données `Graph` : -v
```

Tests des algorithmes de calcul des arbres de recouvrement minimaux et des structures associées:

```
Test de la structure de données `Component` : -v
Test de la structure de données `ConnectedComponents` : -v
Test de l'algorithme de Kruskal : -v
Test de l'algorithme de Prim : -v
```

Algorithme de Rosenkrantz, Stearns et Lewis (RSL)

L'algorithme RSL a été implémenté dans la fonction `rsl`. L'implémentation utilise directement les résultats des algorithmes de calcul des arbres de recouvrement minimaux. La fonction `rsl` prends en entrée deux arguments: le graphe et le noeud racine (optionnel).

L'algorithme de Prim retourne maintenant en sortie l'arbre de recouvrement (MST) et les noeuds du MST en préordre. L'approximation de la tournée minimale est obtenue en connectant les noeuds du MST directement.

La fonction `check_triangular_inequality` permet de vérifier si la fonction de coût des arêtes d'un graphe respecte l'inégalité triangulaire. L'exécution de la fonction est un peu lente. Mais globalement, la plupart des intances de fichiers `stsp` ne respectent pas l'inégalité triangulaire.

La fonction `plot_tsp_rsl_solution` permet de représenter graphiquement l'approximation de la tournée minimale. Seules les instances où les coordonnées des noeuds sont fournies peuvent être représentées sur une figure.

L'algorithme a été testé sur les différentes instances du projet. L'ensemble des résultats se retrouve à la dernière section de ce rapport.

L'exécution est rapide mais la plupart des résultats sont trés imprécis (exemple de l'instance `brg180.tsp`).

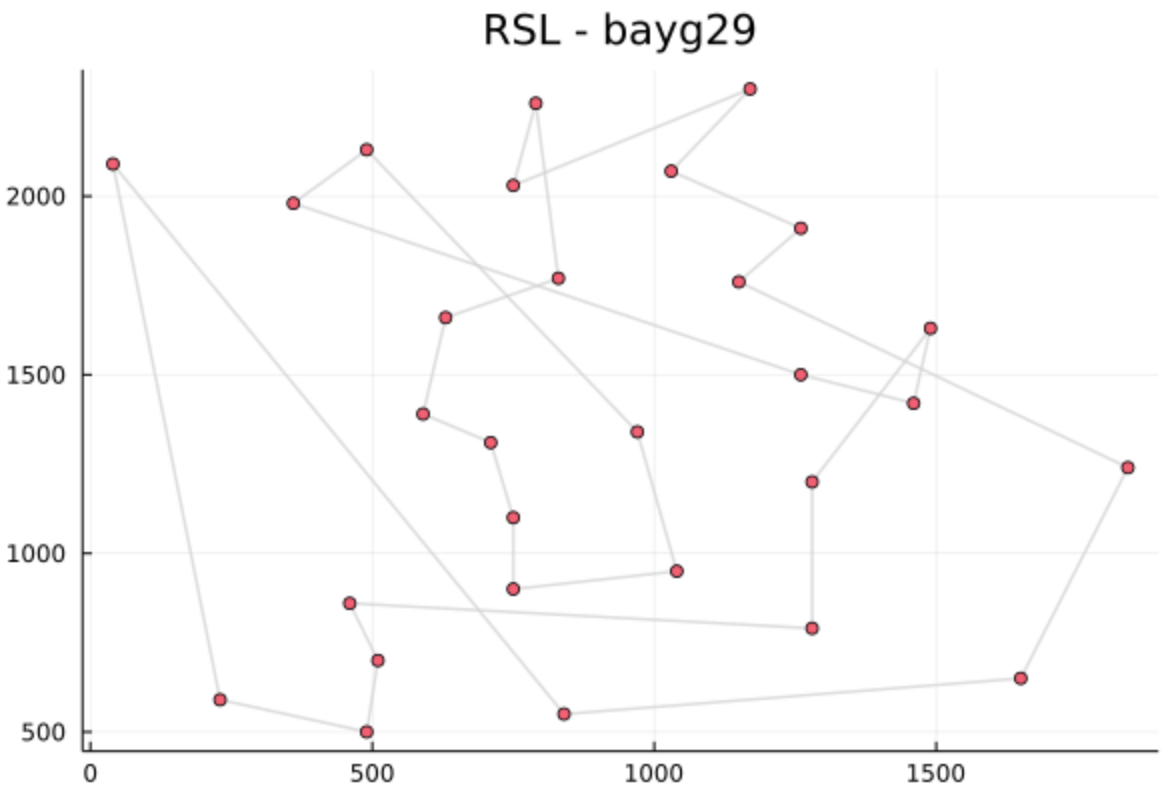
Même en changeant le noeud racine, les résultats changent très peu (voir la deuxième fonction `rsl`).

Exemple de solutions TSP obtenues avec l'algorithme RSL

Instance `bayg29.tsp`

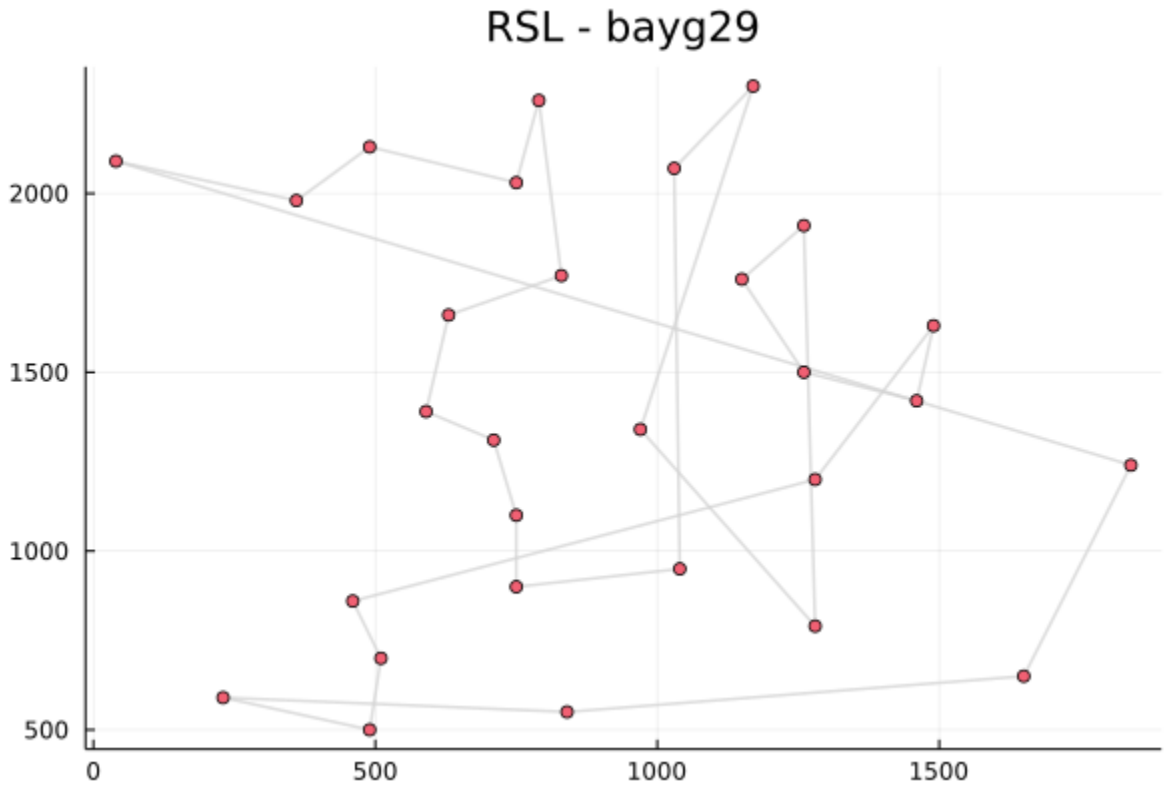
- Taille de la Solution Optimale: 1610
- Taille de la Solution RSL: 2541
- Erreur relative: 57.83%
- Test de l'inégalité triangulaire: True
- Vérification de l'inégalité $tsp_{RSL} \leq 2 \times tsp_{Optimal}$: $2541 \leq 2 \times 1610$ (True)

La solution RSL est représenté sur la figure ci-dessous:


RSL - bayg29
2000
1500
1000
500
0
500
1000
1500

En variant le noeud source et en gardant la solution tsp qui possède la plus faible erreur relative, on peut obtenir une solution TSP de taille 2493 et d'erreur relative 54.84%. On remarque ainsi que l'amélioration est très faible.

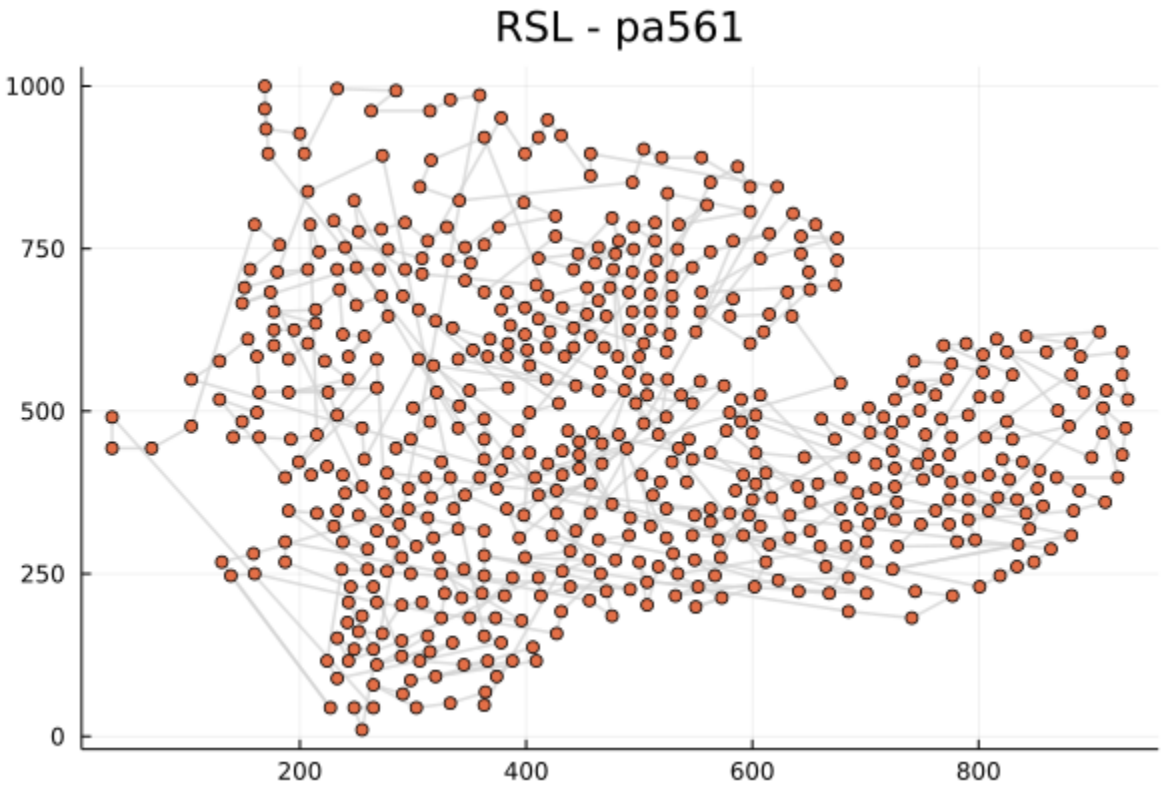
La solution optimisée est représentée sur la figure ci-dessous:



Instance `pa561.tsp`

- Taille de la Solution Optimale: 2763
- Taille de la Solution RSL: 6569
- Erreur relative: 137.75%
- Test de l'inégalité triangulaire: False
- Vérification de l'inégalité $tsp_{RSL} \leq 2 \times tsp_{Optimal}$: $6569 \leq 2 \times 2763$ (False)

La solution RSL est représenté sur la figure ci-dessous:



Instance `brg180.tsp`

- Taille de la Solution Optimale: 1950
- Taille de la Solution RSL: 259290
- Erreur relative: 13196.92%
- Test de l'inégalité triangulaire: False
- Vérification de l'inégalité $tsp_{RSL} \leq 2 \times tsp_{Optimal}$: 259290×1950 (False)

On remarque ainsi que l'approximation de la tournée minimale peut être très mauvaise lorsque l'inégalité triangulaire n'est pas respectée.

Algorithme de Held et Karp (HK)

L'algorithme RSL a été implémenté dans la fonction `hk` . L'implémentation utilise directement **l'algorithme** décrit dans la référence du projet. La fonction `hk` prends en entrée le graphe et plusieurs paramètres optionnels. Les paramètres optionnels sont:

- Le noeud racine.
- L'algorithme de calcul de L'arbre de recouvrement minimal (PRIM ou KRUSKAL)
- La longueur de pas
- Le nombre d'itérations
- Une variable booléenne pour l'affichage des résultats sur le console

L'exécution de l'algorithme s'arrête lorsqu'une tournée minimiale est trouvée ou lorsque le nombre maximal d'itérations est atteint. Le 1_tree ainsi que la borne inférieur de la taille de la tournée minimale sont retournées en sortie.

La fonction `plot_tsp_hk_solution` permet de représenter graphiquement l'approximation de la tournée minimale. Seules les instances où les coordonnées des noeuds sont fournies peuvent être représentées sur une figure.

L'algorithme a été testé sur les différentes instances du projet. L'ensemble des résultats se retrouve à la dernière section de ce rapport.

L'exécution de l'algorithme HK est plus lente que l'éxécution de L'algorithme RSL mais la plupart des résultats sont trés précis.

L'algorithme HK donne des résultats précis même dans les cas où l'inégalité traingulaire n'est pas respectée (exemple de l'instance `brg180.tsp`).

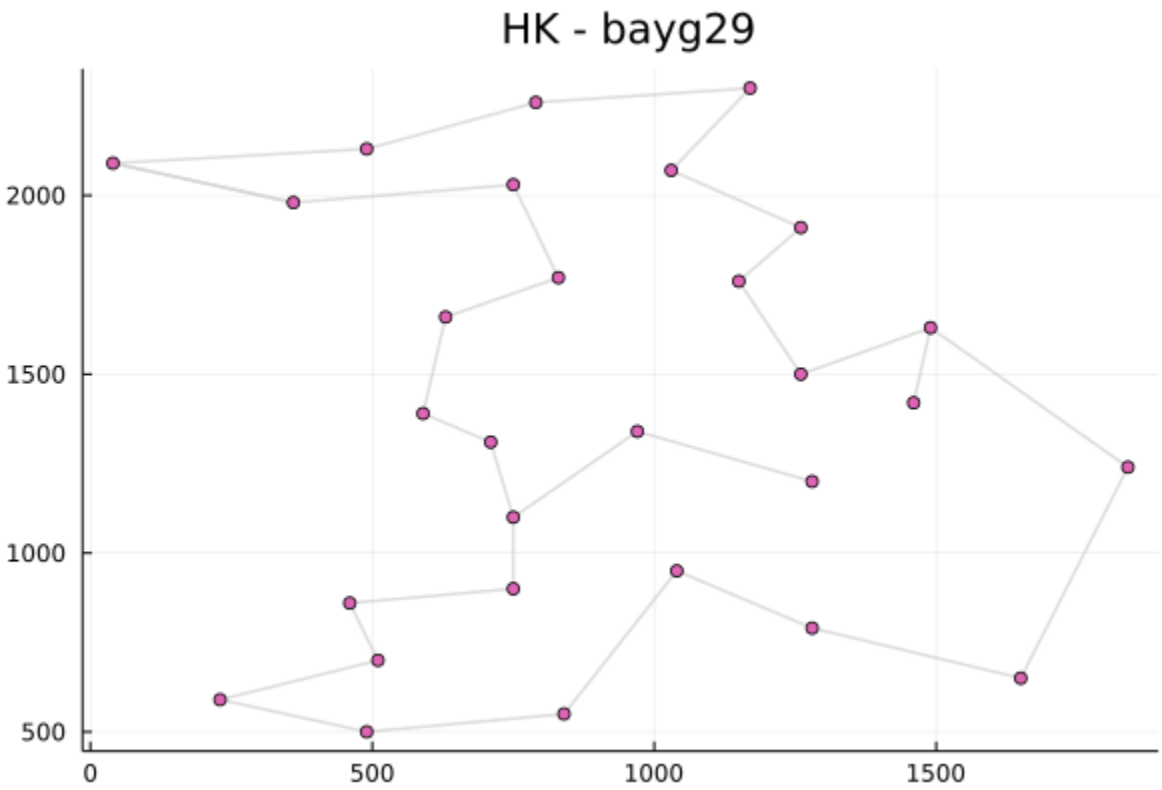
Enfin, l'algorithme HK, comparé à l'algorithme RSL, semble être plus sensible aux paramètres d'entrées (exemple de l'instance `brazil58.tsp`).

Exemple de solutions TSP obtenues avec l'algorithme HK

Instance `bayg29.tsp`

- Taille de la Solution Optimale: 1610
- Taille de la Solution HK: 1607
- Erreur relative: 0.19%
- Noeud racine: noeud d'index 5
- Longeur de pas: 1.0
- Nombre d'itérations: 200
- Algorithme MST: Prim

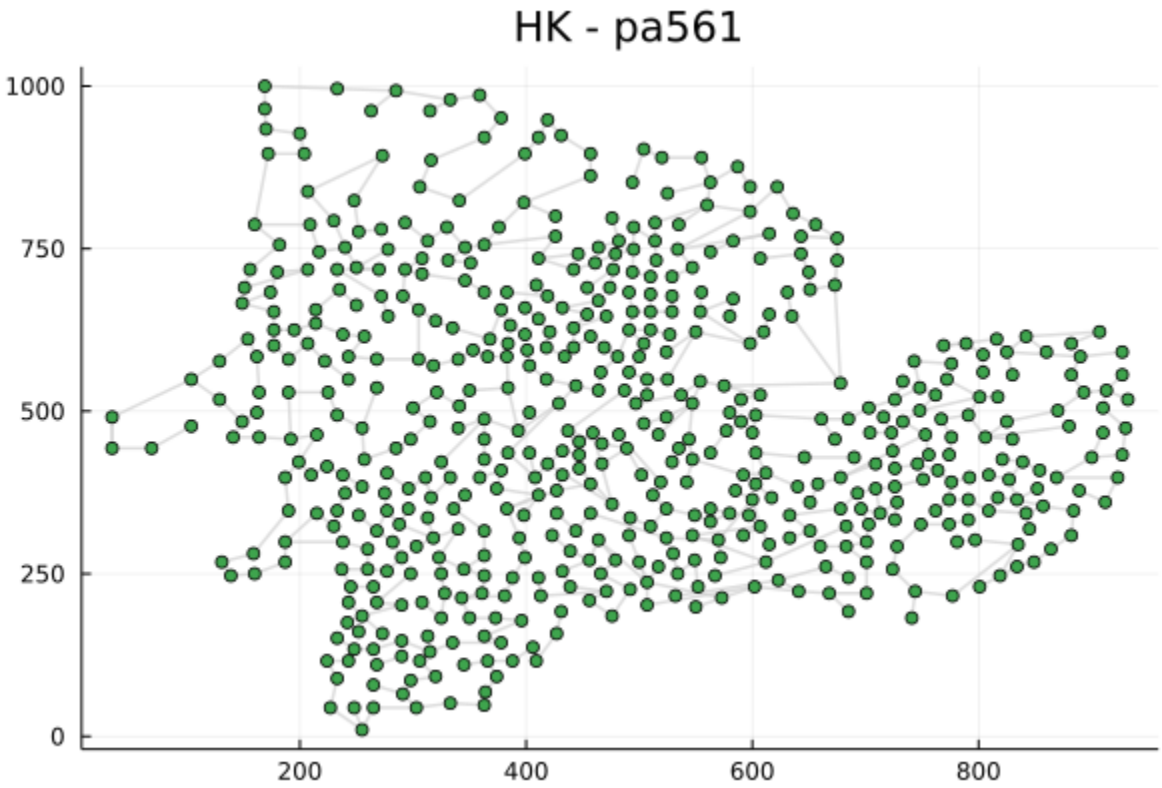
La solution HK est représenté sur la figure ci-dessous:



Instance `pa561.tsp`

- Taille de la Solution Optimale: 2763
- Taille de la Solution HK: 2630
- Erreur relative: 4.81%
- Noeud racine: noeud d'index 258
- Longeur de pas: 1.0
- Nombre d'itérations: 500
- Algorithme MST: Kruskal

La solution HK est représenté sur la figure ci-dessous:



Instance `brg180.tsp`

- Taille de la Solution Optimale: 1950
- Taille de la Solution HK: 1940
- Erreur relative: 0.51%
- Nombre d'itérations: 100
- Algorithme MST: PRIM

Instance `brazil58.tsp`

- Cas 1
  - Taille de la Solution Optimale: 25395
  - Taille de la Solution HK: 19740
  - Erreur relative: 22.27%
  - Noeud racine: noeud d'index 1
  - Nombre d'itérations: 100
  - Algorithme MST: PRIM
- Cas 2
  - Taille de la Solution Optimale: 25395
  - Taille de la Solution HK: 25354
  - Erreur relative: 0.16%
  - Noeud racine: noeud d'index 32
  - Nombre d'itérations: 15000
  - Algorithme MST: KRUSKAL

Programme principal

La fonction *main* permet de lire l'ensemble des fichiers contenus dans le repertoire `instances/stsp/`. Pour chaque fichier, on construit le graphe correspondant et on calcule les arbres de recouvrement minimaux avec les algorithmes de Kruskal et de Prim. Ensuite les tournées minimales sont calculées avec l'algorithme RSL et HK. Les valeurs par défaut des paramètres des différents algorithmes sont utilisées dans toutes les instances.

La fonction `run_tsp_instance` permet d'éxécuter une instance en particulier avec des paramètres spécifiés en entrée.

Résultats du programme principal:

```
File: bayg29.tsp
Reading of header : -v
Reading of nodes : -v
Reading of edges : -v
Arbre de recouvrement minimal : -v
    Kruskl : 1319
    Prim : 1319
TSP solution: -v
    Optimal cycle = 1610
    RSL algorithm:
        RSL cycle weight = 2541
        Relative Error = 57.83%
         2541 ≤ 2×1610 (true)
    HK algorithm:
        HK cycle weight = 1607
        Relative Error = 0.19%
File: bays29.tsp
Reading of header : -v
Reading of nodes : -v
```

Résumé des résultats

Instance	Taille Optimale	Taille RSL (erreur)	Taille HK (erreur)
bayg29	1610	2541 (57.83%)	1607 (0.19%)
bays29	2020	3635 (79.95%)	2006 (0.69%)
brazil58	25395	38939 (53.33%)	19740 (22.27%)
brg180	1950	259290 (13196.92%)	1940 (0.51%)
dantzig42	699	967 (38.34%)	688 (1.57%)
fri26	937	1400 (49.41%)	935 (0.21%)
gr120	6942	15943 (129.66%)	6781 (2.32%)
gr17	2085	2981 (42.97%)	1880 (9.83%)
gr21	2707	4208 (55.45%)	2607 (3.69%)
gr24	1272	2019 (58.73%)	1270 (0.16%)
gr48	5046	10983 (117.66%)	4823 (4.42%)
hk48	11461	18688 (63.06%)	11142 (2.78%)
pa561	2763	6569 (137.75%)	2623 (5.07%)
swiss42	1273	2001 (57.19%)	1269 (0.31%)