Université de Nantes — UFR Sciences et Techniques
Master Informatique parcours “Optimisation en Recherche Opérationnelle (ORO)”
Année académique 2021-2022

Rapport du Devoir Maison n°2

Métaheuristiques

Nicolas COMPÈRE – Adrien CALLICO

November 9, 2021

Livrable du devoir maison 3 :
Battle of metaheuristics

Présentation succincte des choix de mise en œuvre de la métaheuristique concurrente à GRASP appliquée au SPP

La méthode Tabou, ou Tabu Search, est une méthode déterministe de recherche d'optimum. L'idée est de chercher la meilleure solution possible d'un problème (ici un SPP) en parcourant l'ensemble des solutions, sans rester bloqué dans un optimum local comme pour une plus profonde descente. Cela signifie donc avancer dans l'espace des solutions sans pouvoir immédiatement revenir en arrière.

Pour démarrer, l'algorithme part d'une solution admissible et cherche son meilleur voisin. (Cette démarche s'applique même si le voisin trouvé est moins bon que la solution actuelle). Une fois le meilleur voisin trouvé, le mouvement réalisé est enregistré comme mouvement tabou, c'est-à-dire inutilisable pour un certain nombre de tours de boucle. Ce meilleur voisin est ensuite comparé avec la meilleure solution trouvée jusqu'à maintenant, et la meilleure des deux est gardée en mémoire. On se place ensuite sur le meilleur voisin trouvé pour inspecter son voisinage. La même démarche que précédemment sera prise, avec ceci près que le meilleur voisin trouvé ne devra pas être atteint avec un mouvement actuellement tabou. Au fur et à mesure que l'algorithme se déroule, le nombre de mouvement inutilisable augmente. Pourtant, cela pourrait empêcher de revenir sur des solutions qui ont été vu mais qui permettent de passer sur une partie de l'espace des solutions non visitée. Pour cette raison, il existe un temps ou nombre de boucles au bout desquels un mouvement redevient utilisable.

Dans cette méthode, la liste de mouvements Tabou est l'élément essentiel pour permettre de se déplacer dans une grande partie de l'espace des solutions et d'éviter de retomber dans un optimum local déjà visité à la prochaine itération. La meilleure solution sera donc celle retenue lors du parcours de l'espace des solutions, à l'issue d'un nombre d'itérations ou temps limite.

Analyse a priori sur l'influence de la longueur de la liste tabou :

Si la liste taboue est courte :

- Il y a moins d'interdictions (mouvements tabous).
- L'algorithme tend à parcourir de moins grandes distances dans l'espace de recherche. Il explore moins l'espace de recherche.
- Le risque de cycles est plus grand.

A contrario, si la liste est longue, il y'aura plus d'interdictions et le risque de cyclages est réduit, mais on risquera de "louper" plus d'optimum locaux.

Implémentation :

Nous avons gardé la structure de données avec les vecteurs de vecteurs, et repris quelques fonctions du DM1 :

- l'heuristique de construction gloutonne
- la plus profonde descente, où l'on incorpore la notion de mouvement tabou (pour la version 2 du tabou)
- la fonction d'admissibilité

Nous avons implémenté plusieurs versions de la recherche Tabou, avec chacune leurs spécificités :

- Version 1 : Le mouvement considéré est le 1-1 exchange, et nous stockons la liste des mouvements tabou dans une matrice
- Version 2 : Le mouvement considéré est le 2-1 exchange, dont nous explorons qu'un sous-voisinage (aléatoire).
- Version 3 : On applique à chaque itération une plus profonde descente successivement sur 3 voisinages : 2-1 exchange, 1-1 exchange et 0-1 exchange

Pour les versions 2 et 3, afin de ne pas avoir une représentation trop lourde de la liste Tabou, nous avons décidé de stocker les solutions tabous. Ce choix a l'avantage d'éliminer moins de solutions, mais prend un peu plus de place mémoire.

Exemple (instance didactic.dat)

La liste des mouvements tabous est de longueur 3

Une solution admissible de départ est x = [0, 0, 1, 1, 0, 0, 0, 0, 0], avec f(x) = 14)

Détaillons 5 itérations :

tabou V1 : 1-1 échange en plus profonde descente

1ère itération : $x_3 = 0, x_6 = 1$, ce mouvement devient tabou.

sol, f(sol) = ([0, 0, 0, 1, 0, 1, 0, 0, 0], 19)

2e itération : $x_6 = 0, x_2 = 1$, ce mouvement devient tabou.

sol, f(sol) = ([0, 1, 0, 1, 0, 0, 0, 0, 0], 11)

3e itération $x_2 = 0, x_7 = 1$, ce mouvement devient tabou.

sol, f(sol) = ([0, 0, 0, 1, 0, 0, 1, 0, 0], 17)

4e itération : $x_7 = 0, x_3 = 1$, ce mouvement deviens tabou après que le mouvement de la première itération sort de la liste tabou.

sol, f(sol) = ([0, 0, 1, 1, 0, 0, 0, 0, 0], 14)

5e itération : $x_3 = 0, x_2 = 1$, ce mouvement devient tabou après que le mouvement de la deuxième itération sort de la liste tabou.

sol, f(sol) = ([0, 1, 0, 1, 0, 0, 0, 0, 0], 11)

Et ainsi de suite pendant 10 secondes qui est notre critère d'arrêt. On retient ensuite la meilleure solution parmis toutes les itérations.

Expérimentation numérique comparative GRASP vs métaheuristique concurrente

Présentation des instances :

Instance	#Variables	#Contraintes	Densité (%)	Meilleure valeur connue
pb_1000rnd0200	1000	5000	2.59	4
pb_100rnd0100	100	500	2	372
pb_100rnd0300	100	500	2.96	203
pb_100rnd1000	100	300	2	40
pb_200rnd0500	200	1000	2.48	184
pb_200rnd0700	200	200	1.53	1004
pb_200rnd1300	200	600	1.5	571
pb_200rnd1500	200	600	1	926
pb_500rnd0500	500	2500	2.22	122
pb_500rnd0700	500	500	1.2	1141
pb_500rnd0900	500	500	0.9	2236
pb_500rnd1100	500	500	2.26	424

Nos algorithmes ont été implémentés en Julia (version 1.6.0)

L'environnement machine dans lequel nous avons réalisé nos tests présente les caractéristiques suivantes :

Système d'exploitation Ubuntu 20.04

Processeur Intel Core™ i5-7300HQ CPU @ 2.50GHz × 4

Carte Graphique Mesa Intel HD Graphics 630 (KBL GT2)

RAM 8 Go

Le budget de calcul pour toutes les expérimentations est de 10 secondes. Pour chacune des versions, nous avons joué sur la taille de la liste Tabou, ainsi que sur la taille du sous-voisinage

Les résultats de chaque expérimentation sont présentés en annexe.

Discussion

Les résultats des différentes implémentations sont plutôt stables (sauf pour la version 2 lorsque le voisinage est petit) et parviennent parfois à dépasser ceux de GRASP, malgré quelques mauvais résultats, notamment pour l'instance 500_500.

Pour améliorer ces résultats, plusieurs améliorations sont possibles :

- la mise en place d'un critère d'aspiration
- ajouter une composante path-relinking en guise d'intensification
- ajouter un critère d'arrêt en cas de stagnation

L'heuristique de recherche Tabou en l'état (sans ajout de composantes) fournit des résultats assez satisfaisant en comparaison avec GRASP, mais l'utilisation d'une (méta)heuristique à son plein potentiel nécessite de l'expérience et une expertise certaine : on pouvait rapidement régler l'unique

paramètre de GRASP, mais la recherche Tabou nécessite une étude plus approfondie des instances pour pouvoir régler les paramètres ou ajouter une composante supplémentaire.

Annexe

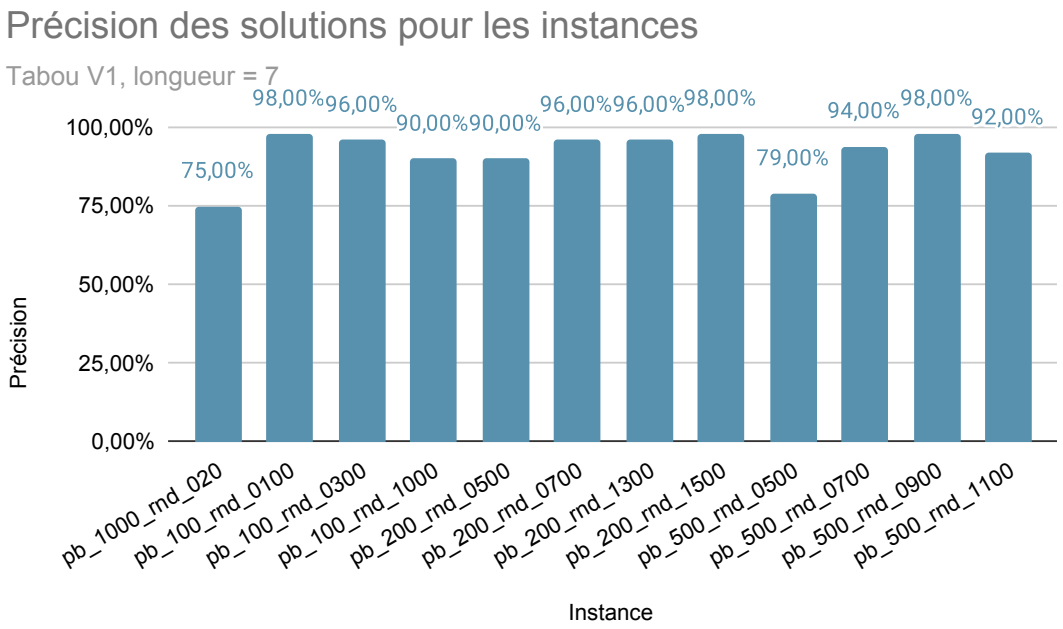

Figure 1: Tabou V1, longueur de 7

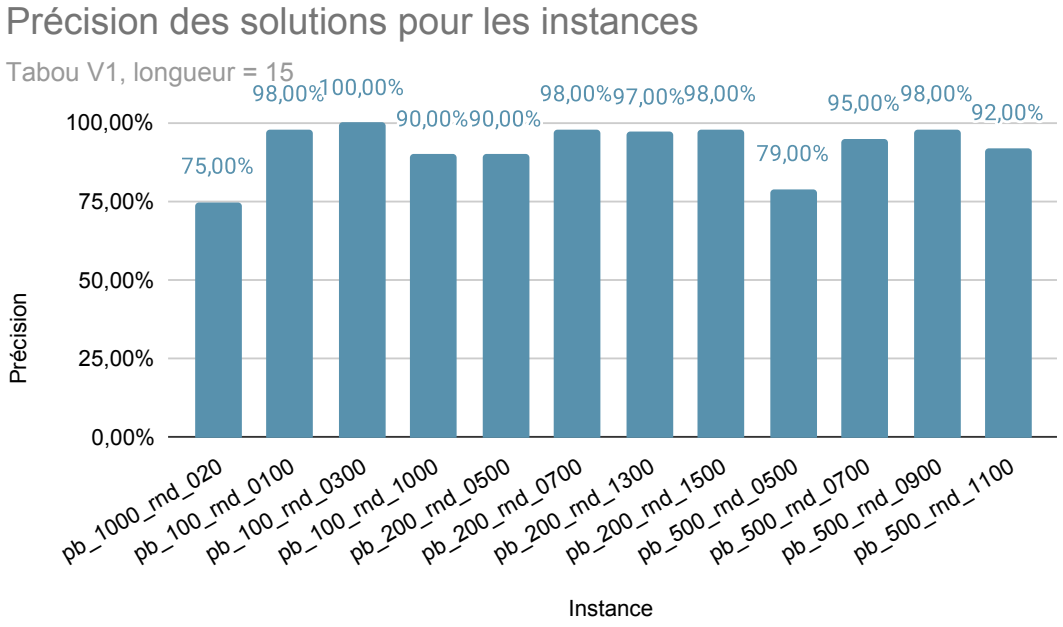

Figure 2: Tabou V1, longueur de 15

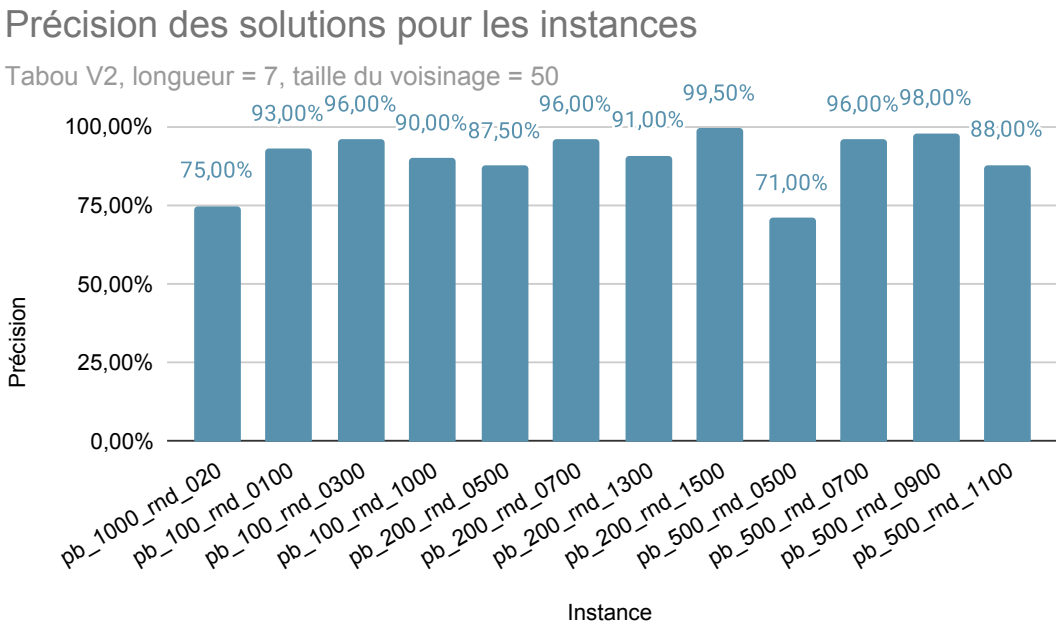

Figure 3: Tabou V2, longueur de 7, voisinage de 50

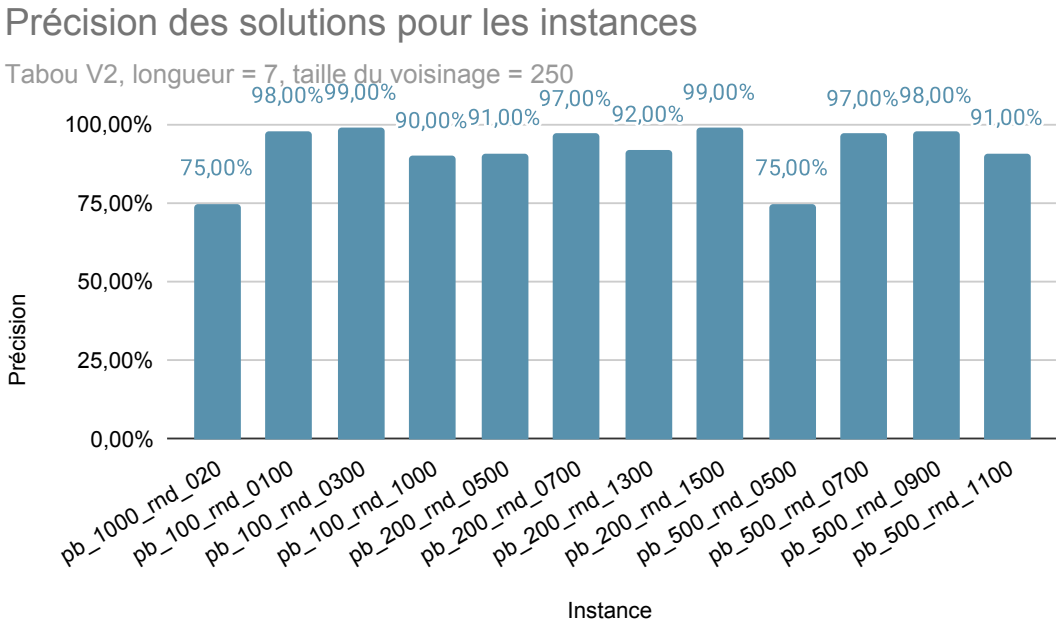

Figure 4: Tabou V2, longueur de 7, voisinage de 250

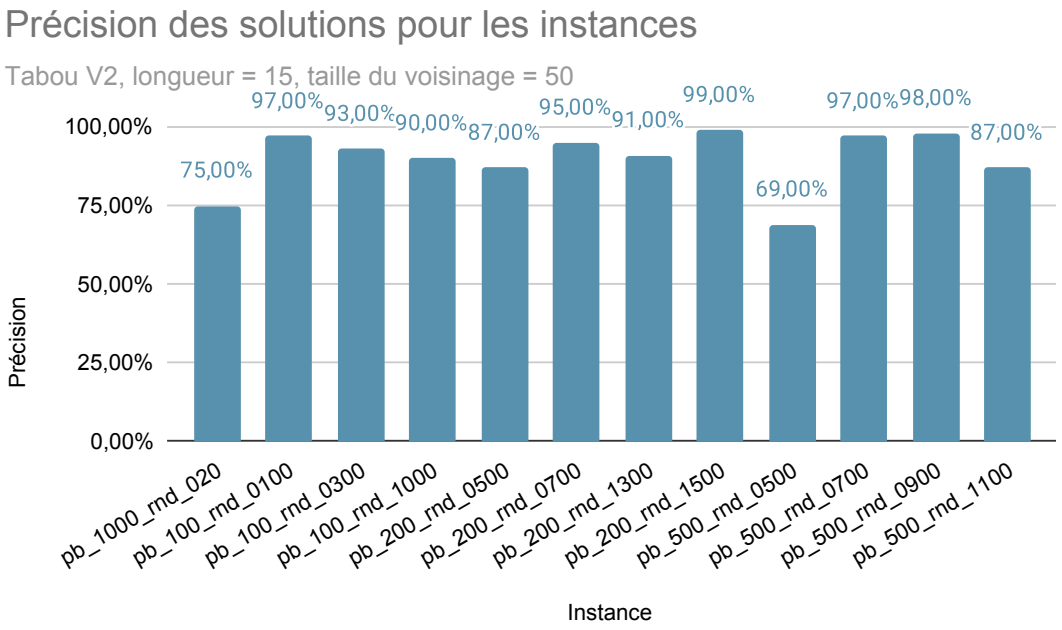

Figure 5: Tabou V2, longueur de 15, voisinage de 50

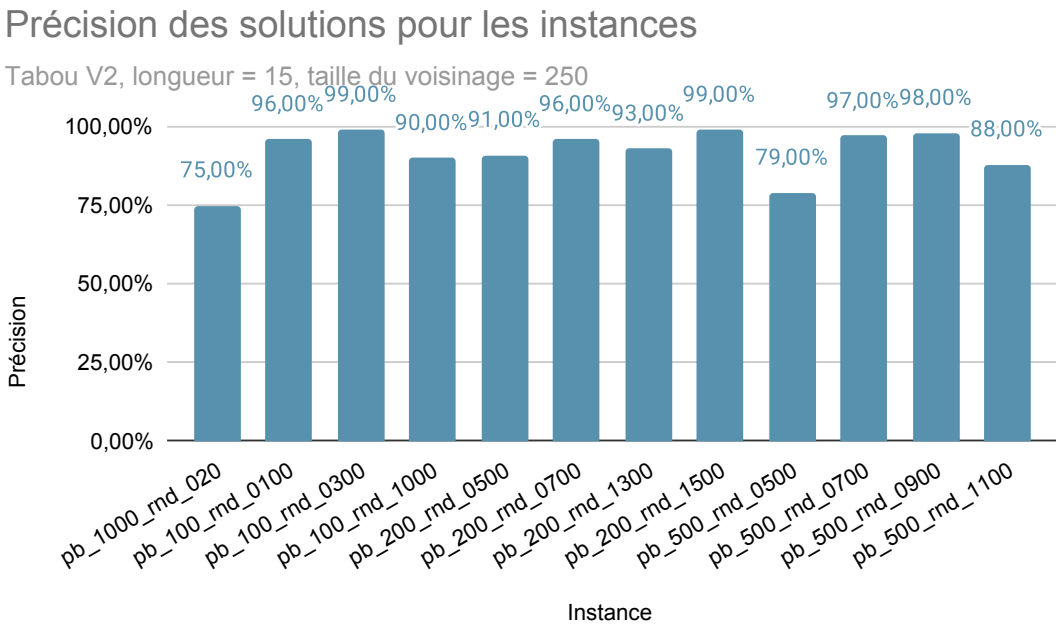

Figure 6: Tabou V2, longueur de 15, voisinage de 250

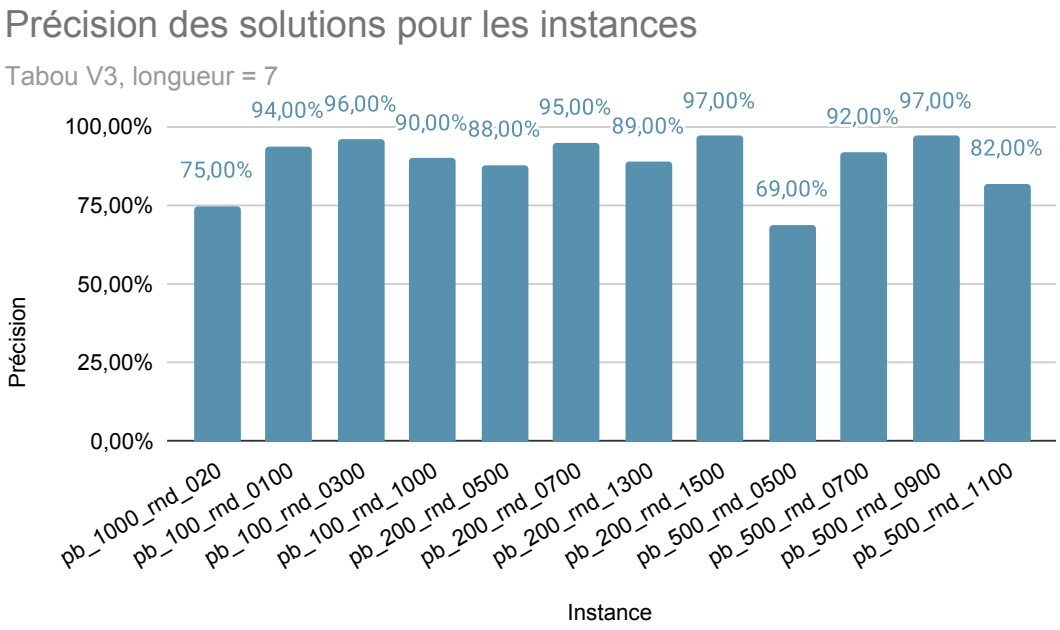

Figure 7: Tabou V3, longueur de 7

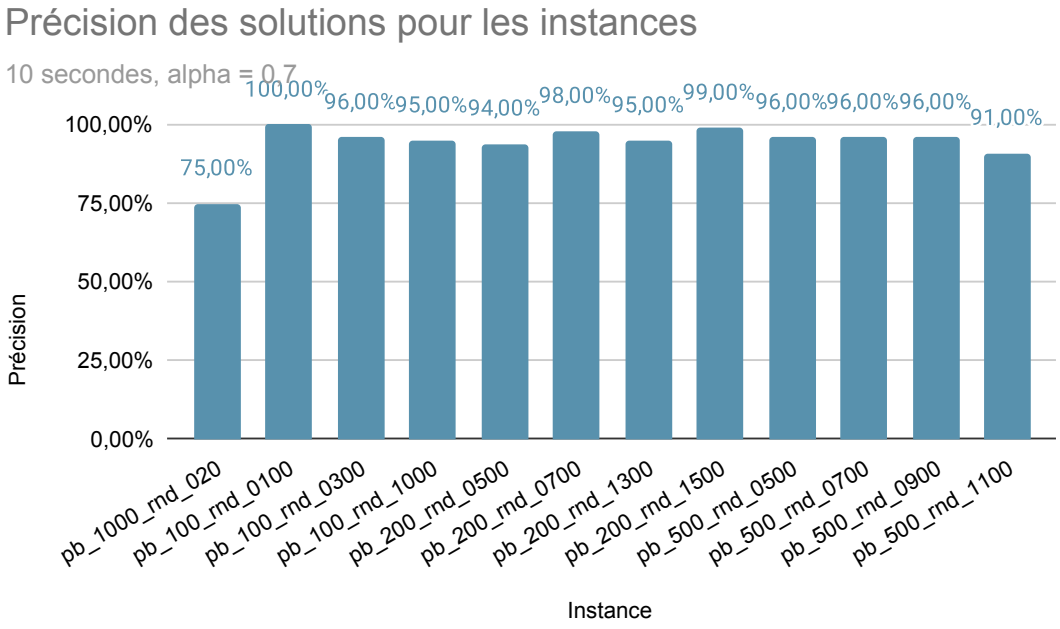

Figure 8: GRASP, $\alpha = 0.7$