# TP : Heuristiques — problème d'ordonnancement

**Q1.1 En utilisant votre langage préféré, écrire un programme qui permet d'évaluer la qualité d'une solution donnée en argument (pour une instance chargée au préalable).**

On a une structure Ordonnancement qui représente une solution possible :

```
struct Ordonancement { //certificat
    vector<Job> jobsSequence; //jobs dans leur ordre d'execution
    vector<int> completion; //temps de completion de chaque tâche

    Ordonancement(vector<Job> j) {
        jobsSequence = j;
    }
    Ordonancement() {}

    void printOrdo() {
        for(int i=0; i<jobsSequence.size(); i++) {
            cout << "job number " << i << ": p = " << jobsSequence[i].p << " w = "
                << jobsSequence[i].w << " d = " << jobsSequence[i].d << endl;
        }
    }
};
```

Et ensuite on a une fonction qui calcule la qualité d'une solution donnée :

```
//Q1.1 ecrire un programme qui permet d'evaluer la qualite d'une solution donnée en argument
int evaluationQualite(Ordonancement o) {
    int qualite = 0;

    //calcul de completion (C dans le TP)
    int c = 0;
    for(int i=0; i<o.jobsSequence.size(); i++) {
        c+=o.jobsSequence[i].p;
        o.completion.push_back(c);
    }

    for(int i=0; i<o.jobsSequence.size(); i++) {
        int T = max((o.completion[i]-o.jobsSequence[i].d), 0);
        qualite+=(T*o.jobsSequence[i].w);
    }
    return qualite;
}
```

**Q1.2 Ecrire un programme qui permet de générer une solution aléatoire et d'évaluer sa qualité.**

Fonction qui crée une solution aléatoire :

```
Ordonancement generationAleatoireDOrdonnancement(Instance instance) {
    vector<Job> sequence;
    int size = instance.jobsList.size();

    while(sequence.size() < size) {
        // on choisit un indice aléatoire qui est compris entre 0 et le nombre de jobs de l'instance:
        int ind = rand() % instance.jobsList.size();
        // on prend le job à l'indice aléatoire:
        Job j = instance.jobsList[ind];
        // on retire ce job de l'instance pour ne pas le réutiliser dans le certificat:
        instance.jobsList.erase (instance.jobsList.begin()+ind);
        sequence.push_back(j);
    }
    Ordonancement o = Ordonancement(sequence);
    return o;
}
```

et ensuite on applique la fonction *evaluationQualite* sur le résultat de la fonction *generationAleatoireDOrdonnancement.*

**Q1.3 Proposer et implémenter une (ou plusieurs) heuristiques constructives**

- Heuristique constructive 1 est basée sur la limite à laquelle l'exécution de la tâche doit être terminée, appelé d. On va les trier par ordre croissant en fonction de d. Cet ordre sera l'ordre d'exécution des tâches.

  ***Comment ça marche ?***
  Notre fonction de tri qu'on applique aux jobs de notre instance :

```
//Q1.3 heuristique constructive 1, tri selon la limite à
//laquelle l'exécution de la tâche doit  être terminée
struct tri_heuristique_constructive_1
{
    inline bool operator() (const Job& job1, const Job& job2)
    {
        return (job1.d < job2.d);
    }
};
```

  Et ensuite on exécute notre fonction pour évaluer la qualité.

- Heuristique constructive 2 est basée sur le ratio entre d et p, noté d/p. On va trier les tâches par ordre croissant en fonction de d/p. Cet ordre sera l'ordre d'exécution.
  Plus précis que Heuristique constructive 1, on se rapproche de l'optimale.

  ***Comment ça marche ?***
  On calcule le ratio d/p pour chaque tâche

```
for (int i = 0; i < listOfJobs1.size();i++) {
    listOfJobs1[i].ratioDelayTimeWeight = listOfJobs1[i].d/listOfJobs1[i].w;
}
```

  Notre fonction de tri qu'on applique à nos jobs de notre instance :

```
//Q1.3 heuristique constructive 2 tri en fonction du ratio d/p
struct tri_heuristique_constructive_2
{
    inline bool operator() (const Job& job1, const Job& job2)
    {
        return (job1.ratioDelayTimeWeight < job2.ratioDelayTimeWeight);
    }
};
```

  Et ensuite on exécute notre fonction pour évaluer la qualité.

- Heuristique constructive 3: on trie les jobs par ordre croissant en fonction de leurs échéances modifiées $\mathbf{modifiedDeadline_j := max\{C + p_j , d_j\}}$ où **C** est la somme des temps d'exécution des jobs déjà ordonnancés, où $\mathbf{p_j}$ est le temps d'exécution et $\mathbf{d_j}$ est la limite à laquelle l'exécution de la tâche doit être terminée.
  Beaucoup plus précis que Heuristique constructive 1 et 2, on se rapproche encore plus de l'optimale et parfois on la trouve comme pour l'instance n40_1 et d'autres.

***Comment ça marche ?***

On retrouve notre fonction permettant de calculer notre **modifiedDeadline$_j$** :

```
//Q1.3 heuristique constructive 3
int modifiedDeadline(int completion, Job job) {
    return max(completion + job.p, job.d);
}
```

Voici notre fonction qui permet de trier les jobs d'une instance en fonction de leurs **modifiedDeadline$_j$ :**

```
vector<Job> sortedJobsBasedOnModifiedDeadline(Instance instance) {
    vector<Job> unsortedJobs = instance.jobsList;
    vector<Job> sortedJobs;
    int completion = 0;
    while (unsortedJobs.size() != 0) {
        Job bestJob = unsortedJobs[0];
        int bestModifiedDeadline = modifiedDeadline(completion,bestJob);
        int indexBestJob = 0;
        for(int i = 0; i< unsortedJobs.size(); i++) {
            int currentModifiedDeadline = modifiedDeadline(completion,unsortedJobs[i]);
            if(currentModifiedDeadline < bestModifiedDeadline) {
                bestModifiedDeadline = currentModifiedDeadline;
                bestJob = unsortedJobs[i];
                indexBestJob = i;
            }
        }
        sortedJobs.push_back(bestJob);
        unsortedJobs.erase(unsortedJobs.begin()+ indexBestJob);
        completion += bestJob.p;
    }
    return sortedJobs;
}
```

**Q1.4 Proposer et implémenter (au moins) un voisinage pour la conception d'heuristiques par recherche locale. En déduire une (ou plusieurs) recherche locale simple de type HillClimbing ou VND. Veillez à spécifier les différents choix de conception (initialisation, type de mouvement, etc).**

Nous avons choisi d'implémenter une recherche locale de HillClimbing, dont voici les choix de conception :
On prend comme solution initiale l'ordonnancement donné par notre heuristique constructive 3 c'est-à-dire les jobs (tâches) triés en fonction de la **modifiedDeadline.**
Ensuite, à partir de cet ordonnancement, on utilise l'inversion pour générer tous les voisins d'une solution. On utilise la recherche locale: Move best improvement pour choisir le meilleur voisin d'une solution.

Et notre condition d'arrêt: lorsqu'on a pas trouvé de solution améliorante.

***Implémentation :***

```
//hill climing methode inversion avec best improvment
Ordonancement localSearchHillClimbing(Instance instance) {
    //Solution initial
    Ordonancement o(instance.jobsList);
    //évaluation de la qualité de l'état initial:
    int eval = evaluationQualite(o);
    int evalInit;

    //création de tous les voisins:
    do {
        Ordonancement init = o;
        evalInit = eval;
        for(int i=0; i<o.jobsSequence.size()-1; i++) {
            Ordonancement o_voisin = init;
            Job job = o_voisin.jobsSequence[i];
            o_voisin.jobsSequence[i] = o_voisin.jobsSequence[i+1];
            o_voisin.jobsSequence[i+1] = job;
            int tempoEval = evaluationQualite(o_voisin);
            if(tempoEval < eval ) {
                eval = tempoEval;
                o = o_voisin;
            }
        }
    }
    while(evalInit != eval);
    return o;
}
```

**Q1.5 Proposer et implémenter une recherche locale itérée, de type ILS. Veillez à spécifier les différents choix de conception (initialisation, recherche locale de base, perturbation, critère d'acceptation, critère d'arrêt).**

**Voici nos choix de conception :**

- Initialisation : la même chose que la recherche locale ci-dessus, c'est à dire la sortie de l'heuristique constructive 3.
- Recherche locale de base : la recherche locale implémentée à la question 1.4 donc hill climbing.
- Perturbation : nous avons choisi de faire un mouvement (swap) aléatoire des jobs à chaque passage dans la boucle while perturbation. Nous avons varié la perturbation dans une plage de trois à douze mouvements (swaps) aléatoires. Cette plage nous permet d'échapper à l'optimale local tout en évitant les désavantages d'une solution complètement aléatoire.
  Voici notre implémentation pour cette perturbation :

```
Ordonancement disruptsScheduling(Ordonancement ordonancement) {

    int randomdisturbances =  (rand() % 10) + 3 ;

    Ordonancement disruptedScheduling = ordonancement;

    for(int i = 0 ; i < randomdisturbances; i++) {
        int indexFirstJob = (rand() % 40);
        int indexSecondJob = (rand() % 40);
        Job job = disruptedScheduling.jobsSequence[indexFirstJob];
        disruptedScheduling.jobsSequence[indexFirstJob] = disruptedScheduling.jobsSequence[indexSecondJob];
        disruptedScheduling.jobsSequence[indexSecondJob] = job;
    }

    return disruptedScheduling;
}
```

- Critère d'acceptation : Nous retenons la solution courante si elle a une meilleure qualité que la précédente.

- Critère d'arrêt : Notre critère d'arrêt est basé sur le temps, nous exécutons notre boucle 3*n secondes et renvoyons la meilleure solution trouvée.
  Cela nous permet de trouver l'optimale dans de nombreux cas.

***Implémentation :***

Voici notre implémentation de cette recherche locale ILS où nous appelons notre perturbation à chaque passage dans la boucle et nous partons donc de cette perturbation pour trouver de meilleurs optimum locaux.

```
//recherche Locale ILS with random pertubation
Ordonancement localSearchILS(Instance initSolution, int optimal, double facteurApproximation) {
    //Solution initiale
    Ordonancement initScheduling = localSearchHillClimbing(initSolution);
    int initEval = evaluationQualite(initScheduling);
    chrono::time_point<std::chrono::system_clock> m_StartTime = chrono::system_clock::now();
    chrono::time_point<std::chrono::system_clock> m_EndTime;

    do
    {
        m_EndTime = chrono::system_clock::now();
        Ordonancement disruptedScheduling = disruptsScheduling(initScheduling);
        Instance newInstance(disruptedScheduling.jobsSequence);
        Ordonancement newOrdonnacement = localSearchHillClimbing(newInstance);
        int tempoEval = evaluationQualite(newOrdonnacement);
        if(tempoEval < initEval ) {
            initEval = tempoEval;
            initScheduling = newOrdonnacement;
        }
    } while(chrono::duration_cast<std::chrono::milliseconds>(m_EndTime - m_StartTime).count()/1000 <3* initScheduling.jobsSequence.size());
    return initScheduling;
}
```

***Campagne d'expérimentation et analyse : Dans une troisième étape, il s'agit de mesurer et d'analyser la qualité de votre heuristique. Pour cela, vous avez à disposition les solutions optimales pour chaque instance. Vous fixerez un temps d'exécution maximal pour chaque exécution (par exemple 3 · n secondes, afin de garder un temps total raisonnable pour la réalisation de votre compagne d'expérimentation).***

***Pour chaque instance, on s'intéresse à l'analyse du temps CPU et le nombre d'évaluations utilisées par l'algorithme pour atteindre une solution avec un facteur d'approximation (1 + ε) de l'optimal avec ε ∈ {0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1}.***

Nous avons choisi de tester notre recherche locale ILS (voir Q1.5), pour ce faire, nous l'avons quelque peu modifié. En effet, nous avons ajouté une structure qui sera la sortie de notre ILS :

```
struct ExperimationResult { //Structure pour la campagne d'expérimentation
    int qualite;
    int nombreEval;
    Ordonancement ordonancement;
    chrono::duration<double> elapsed_seconds;

    ExperimationResult(int q, int evalNumber, Ordonancement ordo, chrono::duration<double> elapsed_time) {
        ordonancement = ordo;
        qualite = q;
        nombreEval = evalNumber;
        elapsed_seconds = elapsed_time;
    }
};
```

Cette structure contient nos critères de comparaison (nombre d'évaluations, temps CPU) permettant d'effectuer notre campagne d'expérimentation et également la meilleure qualité et ordonnancement trouvé.

Nous avons donc rajouté dans l'implémentation de notre ILS :
1) Une condition dans notre boucle while pour la stopper lorsque nous trouvons une solution respectant le facteur d'approximation sinon elle continuera de s'exécuter 3*n secondes.
2) Un compteur permettant de compter le nombre d'évaluations utilisées par l'algorithme pour atteindre une solution respectant le facteur d'approximation.
3) Deux variables chrono permettant de calculer le temps CPU utilisé par l'algorithme pour atteindre une solution respectant le facteur d'approximation.

***Ci-dessous : les résultats de cette campagne d'approximation et un compte rendu de nos observations:***

## 1. avec n40 :

| Fichier de données | Optimal | Facteur d'approximation | Temps CPU en secondes | Qualité | Nombres d'évaluations |
|---|---|---|---|---|---|
| n40_4 | 2094 | 1 + 0 | 0.412111 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.001 | 0.545673 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.005 | 0.489091 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.01 | 0.415734 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.05 | 0.478624 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.1 | 0.48658 | 2094 | 58 |
| n40_4 | 2094 | 1 + 0.5 | 0.476301 | 2094 | 58 |
| n40_4 | 2094 | 1 + 1 | 0.0164339 | 3300 | 1 |
| n40_28 | 15 | 1 + 0 | 0.0110376 | 15 | 1 |
| n40_28 | 15 | 1 + 0.001 | 0.0119117 | 15 | 1 |
| n40_28 | 15 | 1 + 0.005 | 0.00986739 | 15 | 1 |
| n40_28 | 15 | 1 + 0.01 | 0.0117538 | 15 | 1 |
| n40_28 | 15 | 1 + 0.05 | 0.0124439 | 15 | 1 |
| n40_28 | 15 | 1 + 0.1 | 0.0108424 | 15 | 1 |
| n40_28 | 15 | 1 + 0.5 | 0.0110983 | 15 | 1 |
| n40_28 | 15 | 1 + 1 | 0.00960759 | 15 | 1 |
| n40_88 | 10021 | 1 + 0 | 120.049 | 10458 | 4527 |
| n40_88 | 10021 | 1 + 0.001 | 120.011 | 10458 | 4681 |
| n40_88 | 10021 | 1 + 0.005 | 120.065 | 10458 | 4476 |

| | | | | | |
|---|---|---|---|---|---|
| n40_88 | 10021 | 1 + 0.01 | 120.066 | 10458 | 4548 |
| n40_88 | 10021 | 1 + 0.05 | 64.3032 | 10496 | 2657 |
| n40_88 | 10021 | 1 + 0.1 | 17.8467 | 10798 | 730 |
| n40_88 | 10021 | 1 + 0.5 | 0.423383 | 14508 | 16 |
| n40_88 | 10021 | 1 + 1 | 0.10518 | 19614 | 3 |
| n40_67 | 65756 | 1 + 0 | 120.078 | 65874 | 3734 |
| n40_67 | 65756 | 1 + 0.001 | 120.058 | 65874 | 3900 |
| n40_67 | 65756 | 1 + 0.005 | 88.9679 | 66068 | 2786 |
| n40_67 | 65756 | 1 + 0.01 | 62.7314 | 66219 | 1979 |
| n40_67 | 65756 | 1 + 0.05 | 2.78871 | 68384 | 89 |
| n40_67 | 65756 | 1 + 0.1 | 2.7714 | 68384 | 89 |
| n40_67 | 65756 | 1 + 0.5 | 0.0660901 | 74423 | 1 |
| n40_67 | 65756 | 1 + 1 | 0.0642791 | 74423 | 1 |
| n40_78 | 0 | 1 + 0 | 0.0203322 | 0 | 1 |
| n40_78 | 0 | 1 + 0.001 | 0.0178468 | 0 | 1 |
| n40_78 | 0 | 1 + 0.005 | 0.0201923 | 0 | 1 |
| n40_78 | 0 | 1 + 0.01 | 0.0159525 | 0 | 1 |
| n40_78 | 0 | 1 + 0.05 | 0.0179108 | 0 | 1 |
| n40_78 | 0 | 1 + 0.1 | 0.0194934 | 0 | 1 |
| n40_78 | 0 | 1 + 0.5 | 0.019468 | 0 | 1 |
| n40_78 | 0 | 1 + 1 | 0.0182376 | 0 | 1 |
| n40_30 | 98 | 1 + 0 | 0.681801 | 98 | 220 |
| n40_30 | 98 | 1 + 0.001 | 0.669548 | 98 | 220 |
| n40_30 | 98 | 1 + 0.005 | 0.660463 | 98 | 220 |
| n40_30 | 98 | 1 + 0.01 | 0.697086 | 98 | 220 |
| n40_30 | 98 | 1 + 0.05 | 0.671409 | 98 | 220 |
| n40_30 | 98 | 1 + 0.1 | 0.183737 | 105 | 49 |
| n40_30 | 98 | 1 + 0.5 | 0.179036 | 132 | 46 |
| n40_30 | 98 | 1 + 1 | 0.171716 | 132 | 46 |

| | | | | | |
|---|---|---|---|---|---|
| n40_43 | 65134 | 1 + 0 | 33.1437 | 65134 | 793 |
| n40_43 | 65134 | 1 + 0.001 | 21.753 | 65185 | 545 |
| n40_43 | 65134 | 1 + 0.005 | 21.3514 | 65185 | 545 |
| n40_43 | 65134 | 1 + 0.01 | 18.8845 | 65618 | 459 |
| n40_43 | 65134 | 1 + 0.05 | 9.65577 | 68365 | 244 |
| n40_43 | 65134 | 1 + 0.1 | 1.97554 | 70596 | 48 |
| n40_43 | 65134 | 1 + 0.5 | 0.0998657 | 85583 | 1 |
| n40_43 | 65134 | 1 + 1 | 0.10008 | 85583 | 1 |
| n40_100 | 157296 | 1 + 0 | 68.101 | 157296 | 1532 |
| n40_100 | 157296 | 1 + 0.001 | 20.5244 | 157440 | 470 |
| n40_100 | 157296 | 1 + 0.005 | 3.29253 | 157618 | 73 |
| n40_100 | 157296 | 1 + 0.01 | 3.21284 | 157618 | 73 |
| n40_100 | 157296 | 1 + 0.05 | 0.106848 | 164769 | 1 |
| n40_100 | 157296 | 1 + 0.1 | 0.0989406 | 164769 | 1 |
| n40_100 | 157296 | 1 + 0.5 | 0.10481 | 164769 | 1 |
| n40_100 | 157296 | 1 + 1 | 0.0976325 | 164769 | 1 |
| n40_124 | 73041 | 1 + 0 | 40.1923 | 73041 | 952 |
| n40_124 | 73041 | 1 + 0.001 | 39.5017 | 73041 | 952 |
| n40_124 | 73041 | 1 + 0.005 | 31.2777 | 73350 | 718 |
| n40_124 | 73041 | 1 + 0.01 | 4.74692 | 73702 | 116 |
| n40_124 | 73041 | 1 + 0.05 | 1.69671 | 76459 | 38 |
| n40_124 | 73041 | 1 + 0.1 | 0.418722 | 78573 | 7 |
| n40_124 | 73041 | 1 + 0.5 | 0.112721 | 82870 | 1 |
| n40_124 | 73041 | 1 + 1 | 0.118253 | 82870 | 1 |
| n40_111 | 31478 | 1 + 0 | 120.048 | 31874 | 4560 |
| n40_111 | 31478 | 1 + 0.001 | 120.156 | 31874 | 4612 |
| n40_111 | 31478 | 1 + 0.005 | 120.034 | 31874 | 4486 |
| n40_111 | 31478 | 1 + 0.01 | 120.03 | 31874 | 4775 |
| n40_111 | 31478 | 1 + 0.05 | 40.8208 | 33026 | 1515 |

| | | | | | |
|---|---|---|---|---|---|
| n40_111 | 31478 | 1 + 0.1 | 10.9014 | 34493 | 362 |
| n40_111 | 31478 | 1 + 0.5 | 0.0423651 | 45761 | 1 |
| n40_111 | 31478 | 1 + 1 | 0.0424009 | 45761 | 1 |
| n40_5 | 990 | 1 + 0 | 0.456791 | 990 | 114 |
| n40_5 | 990 | 1 + 0.001 | 0.447657 | 990 | 114 |
| n40_5 | 990 | 1 + 0.005 | 0.454006 | 990 | 114 |
| n40_5 | 990 | 1 + 0.01 | 0.359999 | 998 | 88 |
| n40_5 | 990 | 1 + 0.05 | 0.356112 | 998 | 88 |
| n40_5 | 990 | 1 + 0.1 | 0.157987 | 1073 | 31 |
| n40_5 | 990 | 1 + 0.5 | 0.0575572 | 1412 | 10 |
| n40_5 | 990 | 1 + 1 | 0.0582021 | 1412 | 10 |

### 2. avec n50 :

| Fichier de données | Optimal | Facteur d'approximation | Temps CPU en secondes | Qualité | Nombres d'évaluations |
|---|---|---|---|---|---|
| n_50_1 | 2134 | 1 + 0 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.001 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.005 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.01 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.05 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.1 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 0.5 | 150 **(MAX)** | 6250 | 254117 |
| n_50_1 | 2134 | 1 + 1 | 150 **(MAX)** | 6250 | 254117 |
| n_50_26 | 2 | 1 + 0 | 0.0035 | 2 | 1 |
| n_50_27 | 4 | 1 + 0 | 0.0045 | 4 | 1 |
| n_50_25 | 240179 | 1 + 0 | 0.14s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 0.001 | 0.14s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 0.005 | 0.14s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 0.01 | 0.14s | 240179 | 2 |

| | | | | | |
|---|---|---|---|---|---|
| n_50_25 | 240179 | 1 + 0.05 | 0.15s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 0.1 | 0.15s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 0.5 | 0.14s | 240179 | 2 |
| n_50_25 | 240179 | 1 + 1 | 0.13 | 240572 | 1 |
| n_50_125 | 110392 | 1 + 0 | 106.96 | 110392 | 6825 |
| n_50_125 | 110392 | 1 + 0.001 | 52.87 | 110482 | 3375 |
| n_50_125 | 110392 | 1 + 0.005 | 17.82 | 110805 | 1078 |
| n_50_125 | 110392 | 1 + 0.01 | 9.37 | 111477 | 334 |
| n_50_125 | 110392 | 1 + 0.05 | 0.27 | 115483 | 15 |
| n_50_125 | 110392 | 1 + 0.1 | 0.07 | 119055 | 4 |
| n_50_125 | 110392 | 1 + 0.5 | 0.03 | 124051 | 1 |
| n_50_125 | 110392 | 1 + 1 | 0.03 | 124051 | 1 |
| n50_108 | 0 | 1 + 0 | 0.01 | 0 | 1 |
| n50_100 | 120108 | 1 + 0 | 71.72 | 120108 | 3506 |
| n50_100 | 120108 | 1 + 0.001 | 14.59 | 120153 | 744 |
| n50_100 | 120108 | 1 + 0.005 | 1.75 | 120694 | 84 |
| n50_100 | 120108 | 1 + 0.01 | 1.68 | 121298 | 51 |
| n50_100 | 120108 | 1 + 0.05 | 0.115 | 125061 | 3 |
| n50_100 | 120108 | 1 + 0.1 | 0.112 | 126816 | 2 |
| n50_100 | 120108 | 1 + 0.5 | 0.05 | 138014 | 1 |
| n50_100 | 120108 | 1 + 1 | 0.06 | 138014 | 1 |
| n50_96 | 177909 | 1 + 0 | 113.42 | 178028 | 6158 |
| n50_96 | 177909 | 1 + 0.001 | 6.15 | 178028 | 319 |
| n50_96 | 177909 | 1 + 0.005 | 1.83 | 178763 | 97 |
| n50_96 | 177909 | 1 + 0.01 | 0.2 | 179303 | 9 |
| n50_96 | 177909 | 1 + 0.05 | 0.12 | 183793 | 2 |
| n50_96 | 177909 | 1 + 0.1 | 0.07 | 187991 | 1 |
| n50_96 | 177909 | 1 + 0.5 | 0.05 | 187991 | 1 |
| n50_96 | 177909 | 1 + 1 | 0.05 | 187991 | 1 |
| n50_50 | 198076 | 1 + 0 | 0.26 | 198076 | 2 |
| n50_50 | 198076 | 1 + 0.001 | 0.11 | 198094 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| n50_50 | 198076 | 1 + 0.005 | 0.11 | 198094 | 1 |
| n50_50 | 198076 | 1 + 0.01 | 0.11 | 198094 | 1 |
| n50_50 | 198076 | 1 + 0.05 | 0.11 | 198094 | 1 |
| n50_50 | 198076 | 1 + 0.1 | 0.11 | 198094 | 1 |
| n50_50 | 198076 | 1 + 0.5 | 0.11 | 198094 | 1 |
| n50_50 | 198076 | 1 + 1 | 0.11 | 198094 | 1 |
| n50_75 | 135677 | 1 + 0 | 12.50 | 135677 | 504 |
| n50_75 | 135677 | 1 + 0.001 | 11.76 | 135792 | 358 |
| n50_75 | 135677 | 1 + 0.005 | 10.35 | 136284 | 290 |
| n50_75 | 135677 | 1 + 0.01 | 1.59 | 136853 | 71 |
| n50_75 | 135677 | 1 + 0.05 | 0.22 | 141368 | 5 |
| n50_75 | 135677 | 1 + 0.1 | 0.11 | 146837 | 2 |
| n50_75 | 135677 | 1 + 0.5 | 0.06 | 149492 | 1 |
| n50_75 | 135677 | 1 + 1 | 0.06 | 149492 | 1 |
| n50_40 | 26423 | 1 + 0 | 150 **(MAX)** | 26569 | 24171 |
| n50_40 | 26423 | 1 + 0.001 | 150 **(MAX)** | 26569 | 24171 |
| n50_40 | 26423 | 1 + 0.005 | 149,92 | 26569 | 24699 |
| n50_40 | 26423 | 1 + 0.01 | 74.03 | 26643 | 11877 |
| n50_40 | 26423 | 1 + 0.05 | 5.68 | 27471 | 813 |
| n50_40 | 26423 | 1 + 0.1 | 3.19 | 28482 | 444 |
| n50_40 | 26423 | 1 + 0.5 | 0.06 | 38686 | 7 |
| n50_40 | 26423 | 1 + 1 | 0.023 | 42986 | 1 |

### 3. avec n100 :

| Fichier de données | Optimal | Facteur d'approximation | Temps CPU en secondes | Qualité | Nombres d'évaluations |
|---|---|---|---|---|---|
| n100_27 | 718 | 1 + 0 | 300.003s | 1671 | 163479 |
| n100_27 | 718 | 1 + 0.001 | 299.976s | 1671 | 167725 |
| n100_27 | 718 | 1 + 0.005 | 300.003s | 1671 | 177503 |
| n100_27 | 718 | 1 + 0.01 | 300.002s | 1671 | 176900 |

| | | | | | |
|---|---|---|---|---|---|
| n100_27 | 718 | 1 + 0.05 | 300.006s | 1671 | 179637 |
| n100_27 | 718 | 1 + 0.1 | 300.004s | 1671 | 177057 |
| n100_27 | 718 | 1 + 0.5 | 300.008s | 1671 | 176321 |
| n100_27 | 718 | 1 + 1 | 299.968s | 1671 | 177687 |
| n100_49 | 656693 | 1 + 0 | 301.035s | 686188 | 1702 |
| n100_49 | 656693 | 1 + 0.001 | 301.033s | 686188 | 1701 |
| n100_49 | 656693 | 1 + 0.005 | 300.882s | 686188 | 1696 |
| n100_49 | 656693 | 1 + 0.01 | 301.184s | 686188 | 1719 |
| n100_49 | 656693 | 1 + 0.05 | 2.8613s | 687777 | 8 |
| n100_49 | 656693 | 1 + 0.1 | 1.25052s | 691165 | 1 |
| n100_49 | 656693 | 1 + 0.5 | 1.15662s | 691165 | 1 |
| n100_49 | 656693 | 1 + 1 | 1.19326s | 691165 | 1 |
| n100_109 | 232 | 1 + 0 | 0.128228s | 232 | 1 |
| n100_109 | 232 | 1 + 0.001 | 0.122667s | 232 | 1 |
| n100_109 | 232 | 1 + 0.005 | 0.12586s | 232 | 1 |
| n100_109 | 232 | 1 + 0.01 | 0.124951s | 232 | 1 |
| n100_109 | 232 | 1 + 0.05 | 0.125883s | 232 | 1 |
| n100_109 | 232 | 1 + 0.1 | 0.125174s | 232 | 1 |
| n100_109 | 232 | 1 + 0.5 | 0.129434s | 232 | 1 |
| n100_109 | 232 | 1 + 1 | 0.127786s | 232 | 1 |
| n100_81 | 1400 | 1 + 0 | 300.033s | 2199 | 8398 |
| n100_81 | 1400 | 1 + 0.001 | 300.129s | 2199 | 8092 |
| n100_81 | 1400 | 1 + 0.005 | 300.055s | 2199 | 8241 |
| n100_81 | 1400 | 1 + 0.01 | 300.043s | 2199 | 8268 |
| n100_81 | 1400 | 1 + 0.05 | 300.059s | 2199 | 8248 |
| n100_81 | 1400 | 1 + 0.1 | 300.041s | 2199 | 8231 |
| n100_81 | 1400 | 1 + 0.5 | 300.069s | 2199 | 8082 |
| n100_81 | 1400 | 1 + 1 | 0.106258s | 2199 | 1 |
| n100_119 | 293571 | 1 + 0 | 300.35s | 393917 | 2287 |

| | | | | | |
|---|---|---|---|---|---|
| n100_119 | 293571 | 1 + 0.001 | 300.298s | 393917 | 2315 |
| n100_119 | 293571 | 1 + 0.005 | 300.322s | 393917 | 2206 |
| n100_119 | 293571 | 1 + 0.01 | 300.391s | 393917 | 2324 |
| n100_119 | 293571 | 1 + 0.05 | 300.443s | 393917 | 2257 |
| n100_119 | 293571 | 1 + 0.1 | 300.461s | 393917 | 2323 |
| n100_119 | 293571 | 1 + 0.5 | 0.467022s | 402995 | 1 |
| n100_119 | 293571 | 1 + 1 | 0.471903s | 402995 | 1 |
| n100_16 | 407703 | 1 + 0 | 300.797s | 450022 | 5677 |
| n100_16 | 407703 | 1 + 0.001 | 300.817s | 450022 | 5749 |
| n100_16 | 407703 | 1 + 0.005 | 300.948s | 450022 | 5754 |
| n100_16 | 407703 | 1 + 0.01 | 300.808s | 450022 | 5555 |
| n100_16 | 407703 | 1 + 0.05 | 300.798s | 450022 | 5685 |
| n100_16 | 407703 | 1 + 0.1 | 300.887s | 450022 | 5690 |
| n100_16 | 407703 | 1 + 0.5 | 0.811714s | 451064 | 1 |
| n100_16 | 407703 | 1 + 1 | 0.893215s | 451064 | 1 |
| n100_37 | 181850 | 1 + 0 | 300.239s | 262284 | 9551 |
| n100_37 | 181850 | 1 + 0.001 | 300.236s | 262284 | 9807 |
| n100_37 | 181850 | 1 + 0.005 | 300.225s | 262284 | 9806 |
| n100_37 | 181850 | 1 + 0.01 | 300.187s | 262284 | 9803 |
| n100_37 | 181850 | 1 + 0.05 | 300.216s | 262284 | 9339 |
| n100_37 | 181850 | 1 + 0.1 | 300.259s | 262284 | 9687 |
| n100_37 | 181850 | 1 + 0.5 | 0.225504s | 263117 | 1 |
| n100_37 | 181850 | 1 + 1 | 0.227641s | 263117 | 1 |

***Conclusion***

Une première observation relevée lors de nos développements/tests des différents heuristiques est que les heuristiques gloutonnes sont moins performantes (en terme de recherche d'optimisation) comparé aux heuristiques de recherche locale ou l'ILS qui sont plus performantes comparés à l'algorithme utilisant Hill Climbing.

D’après les tests on observe que plus le facteur d’approximation est petit, plus il y a d'évaluations, plus on se rapproche de la valeur optimale et plus le temps CPU est grand. Quand le facteur d’approximation est plus grand, le temps CPU et le nombre d'évaluations se réduisent mais on s'éloigne de la valeur optimale.

On observe également que plus la valeur optimale est petite, plus on arrive à l’obtenir ou à s'en rapprocher facilement. Moins il y a de données, plus on a de chance d'obtenir la valeur optimale.

En conclusion, notre algorithme ILS est efficace avec un facteur d’approximation petit et est très efficace lorsque la valeur optimale est petite et/ou quand le facteur d’approximation est grand. Plus on augmente le temps d'exécution de notre algo, plus on a de chance de se rapprocher de l’optimale et/ou de la trouver.