RAPPORT TECHNIQUE

Extension d'un système d'allocation de ressources

https://github.com/agandois/ResourceAllocator

Goron STEVEN
Gandois ALVIN

Tuteur : M. VANHÉE

M1 Informatique Année 2019 - 2020

Table des matières

1 Introduction

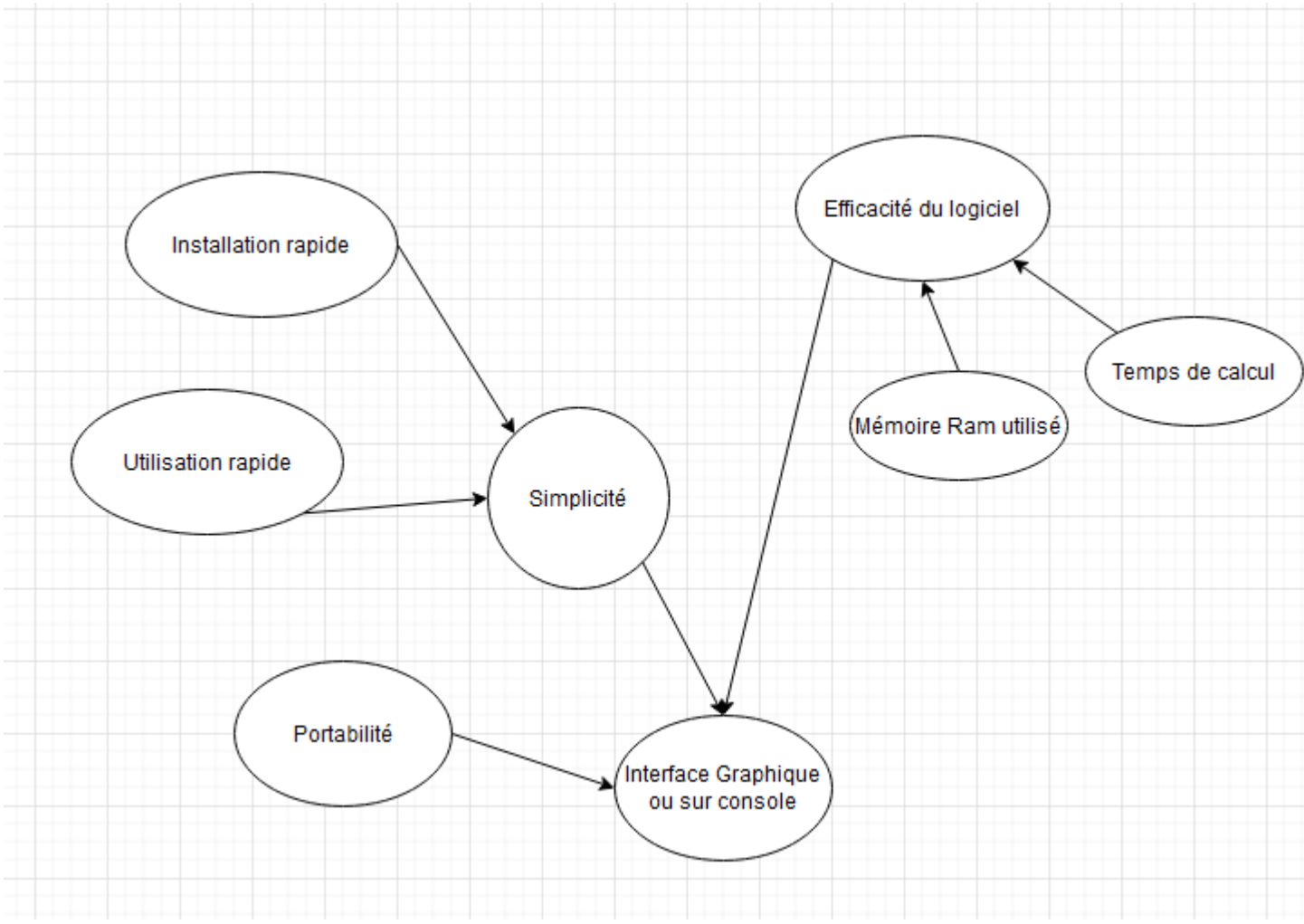
Nous avons choisi ce projet pour les perspectives de développement et de connaissance qu'il nous apporte. Ainsi le projet porte sur les allocations de ressources, une discipline dont nous aurons besoin en tant que futur ingénieur en Intelligence Artificiel. C'est également très utilisé dans le domaine publique, l'exemple le plus connu étant l'affectation de voeux Parcoursup(anciennement APB). C'est d'ailleurs ce système qui a été utilisé pour nous allouer ce projet.

Dans notre cas, nous avons besoin de tels systèmes pour affecter des projets annuels aux étudiants. Un certain nombre de projets sont proposés par des enseignants et les étudiants doivent les classer selon leur préférence. L'objectif est que l'allocation obtenue minimise l'insatisfaction globale des étudiants.

Une solution a déjà été proposée par Lois Vanhée, tuteur de ce projet. Cependant, le programme utilise une bibliothèque payante (gratuite dans un cadre universitaire) et nécessite une installation spécifique, voire parfois difficile, sur le poste de travail.

Notre objectif est alors de reprendre cette solution et de remplacer cette bibliothèque par une bibliothèque gratuite et libre d'utilisation et qui soit également simple à utiliser, pour qu'elle puisse être embarquée directement dans l'application sans ne nécessiter aucune action supplémentaire de l'utilisateur final.

Nous voulons que l'application ait les caractéristiques suivantes :



Les principaux défis techniques spécifiques du projet sont l'utilisation de la programmation

linéaire, premièrement de trouver une bibliothèque à la hauteur de nos attentes, puis ensuite de reprogrammer les différentes méthodes utilisant la bibliothèque originale.

2 Contexte

> En informatique, plus précisément en recherche opérationnelle et d'optimisation combinatoire, le problème d'affectation consiste à attribuer au mieux des tâches à des agents. Chaque agent peut réaliser une unique tâche pour un coût donné et chaque tâche doit être réalisée par un unique agent. Les affectations (c'est-à-dire les couples agent-tâche) ont toutes un coût défini. Le but est de minimiser le coût total des affectations afin de réaliser toutes les tâches.

(Wikipedia)

Ce problème ne doit pas être confondu avec le problème des mariages stables. En effet, bien que l'exemple cité en introduction (Parcoursup/APB) soit en fait un problème de mariage stable, dans notre cas seuls les agents classent les tâches selon leur préférence, tandis que le problème de mariage stable nécessite que les tâches aient également un ordre de préférence des candidats.

3 État de l'art

En informatique, le problème d'affectation peut être réduit à un problème de couplage parfait dans un graphe biparti. Cependant, dans notre cas, certains projets (ou tâches) peuvent être affectés à plusieurs étudiants. Par exemple, un enseignant peut proposer un même projet à deux étudiants différents, donc nous ne sommes plus dans le cas d'un couplage.

La solution fournie utilise la programmation linéaire et nous avons décidé de garder cette méthode qui permet de résoudre le problème en un temps raisonnable.

4 Outils existants

4.1 CPLEX

Cplex est la bibliothèque de programmation linéaire développée par IBM, utilisée dans la solution proposée par M. Vanhée. Bien que très performante, cette bibliothèque est payante (sauf dans un cadre universitaire), et nécessite une installation spécifique. Dans notre cas, malgré le fait que nous sommes étudiants, nous avons eu des difficultés lors de l'installation de Cplex : pour nous inscrire, il fallait utiliser nos mails étudiant, mais l'université n'était alors pas reconnue comme établissement universitaire par le site de Cplex. C'est uniquement après quelques jours que l'université a finalement été reconnue et que l'on puisse enfin s'inscrire et installer le logiciel.

Cplex permet de modélisez mathématiquement des problèmes et de les résolvez avec les algorithmes puissants de CPLEX Optimizer, qui peuvent produire des décisions précises et logiques. La technologie de programmation mathématique de CPLEX Optimizer permet une optimisation des décisions pour améliorer l'efficacité, réduire les coûts et augmenter la rentabilité.

CPLEX Optimizer fournit des solveurs de programmation mathématique flexibles et hautes performances pour les problèmes de programmation linéaire, de programmation mixte en nombres entiers, de programmation par contraintes et de programmation à contraintes quadratiques. Ces solveurs incluent un algorithme parallèle distribué pour la programmation mixte en nombres entiers afin de tirer parti de plusieurs ordinateurs pour résoudre des problèmes difficiles.

Ainsi Cplex peut être utiliser de différentes maniére tel que :

— Quel est le meilleur plan pour mon usine afin de répondre à la demande de produits finis, tout en minimisant les coûts de configuration machine et en tenant compte des arrivées planifiées de matières premières ?
— Comment puis-je affecter de façon optimale des campagnes marketing à des clients en tenant compte des prévisions sur la propension des clients à répondre, et maximiser les achats attendus, tout en respectant les contraintes budgétaires ?

4.2 Or tools

Or-tools, développpé par Google, est également une bibliothèque de programmation linéaire. L'ayant utiliser en cours de programmation linéaire, nous pensions à l'utiliser pour ce projet. Celle-ci est gratuite, cependant elle nécessite une installation spécifique et donc ce n'est pas optimale pour répondre à nos objectifs. Nous avons alors cherché une autre bibliothèque.

Voici quelques exemples de problèmes que OR-Tools résout :

— Acheminement des véhicules : trouvez des itinéraires optimaux pour les flottes de véhicules qui ramassent et livrez des colis en fonction des contraintes (par exemple, "ce camion ne peut pas contenir plus de 20000 livres" ou "toutes les livraisons doivent être effectuées dans un délai de deux heures ").
— Planification : recherchez la planification optimale pour un ensemble complexe de tâches, dont certaines doivent être effectuées avant d'autres, sur un ensemble fixe de machines ou d'autres ressources.
— Emballage des bacs : Emballez autant d'objets de différentes tailles que possible dans un nombre fixe de bacs avec des capacités maximales.

4.3 SCPSolver

SCPSolver est une interface de programmation linéaire qui implémente plusieurs bibliothèques : Lpsolve, GLPK, Cplex. Elle a l'avantage d'être très simple à installer : il suffit de télécharger la bibliothèque ainsi qu'un solveur proposé.

Les objectifs principaux de SCPsolver sont :

— faciliter le développement
— garder l' API learn
— devenir "indépendant de la plateforme"
— automatiser le déploiement de la bibliothèque binaire
— modélisation séparée du solveur

5 Outils utilisés

5.1 Java

Java est un langage de programmation orienté objet qui a l'avantage d'être portable, c'est à dire que notre application fonctionnera quelque soit l'environnement dans lequel elle est utilisée. C'est le langage utilisé par la solution proposée initialement, et nous avons gardé ce langage car il répond à un de nos objectifs, qui est la portabilité.

Notre groupe ayant déjà des connaissances en java, lorsque nous avons appris que ce projet était réalisé en Java, nous nous sommes dit que nous avions des avantages considérable vis-à-vis des autres langages de programmation même si nous avions encore quelque lacune sur certain point.

5.2 SCPSolver

SCPSolver est une bibliothèque de modélisation et de résolution de programmes linéaires. Cette bibliothèque est une interface implémentant différentes bibliothèques de programmation linéaire telles que LPSolve ou GLPK.

5.3 Eclipse

Eclipse est un environnement de développement intégré utilisé dans la programmation informatique. Il contient un espace de travail de base et un système de plug-in extensible pour personnaliser l'environnement comme nous en avions besoin pour utiliser Cplex. Eclipse est écrit principalement en Java et son utilisation principale est pour le développement d'applications Java. Il peut également être utilisé pour développer des documents avec LaTeX et des packages . Les environnements de développement incluent les outils de développement Java Eclipse (JDT) pour Java et Scala, Eclipse CDT pour C / C ++ et Eclipse PDT pour PHP, entre autres.

Nous avons choisi cet outil pour développer notre application principalement car c'est ce qui a été utilisé pour développer l'application d'origine. De plus, il existe beaucoup de documentations sur internet, ce qui nous a permis de résoudre facilement nos problèmes lié a l'installation de Cplex. Il permet de modifier facilement les configurations de la JVM. Il nous permet de créer des jar et d'utiliser les jar de SCPSolver de manière simple.

6 Réalisation technique

L'utilisation de SCPSolver diffère grandement de Cplex. Premièrement, nous ne pouvons pas nommer nos variables : lorsque l'on créé une variable, un identifiant lui est associé (de 0 à n). Pour palier à ce problème, nous avons défini une map qui associe le nom d'une variable à son identifiant. Ainsi, à chaque fois que l'on créé une variable, nous ajoutons cette variable dans notre map.

```
static Map<String , Integer> NameToId = new HashMap<>();
```

FIGURE 1 – Map associant un ID de variable à son nom

```
private static void addVar(String name) {
    if (!NameToId.containsKey(name))
        NameToId.put(name, NameToId.size());
}
```

FIGURE 2 – Méthode permettant d'ajouter une variable à notre map

6.1 Fonction objectif

La deuxième différence est que TOUTES les variables doivent être définies AVANT l'instanciation du solveur. En effet, lors de l'instanciation du solveur, nous devons définir les poids de nos variables dans la fonction objectif. Par exemple, si nous avons deux variables et que la fonction objectif est

$3x + 1y >= 8$

Alors nous définissions un tableau de double content les valeurs 3 et 1, et on instancie notre solveur avec ce tableau. Ainsi, non seulement toutes les variables doivent être définies **avant** l'instanciation, mais aussi la fonction objectif !

La fonction objectif est la suivante :

$$min \sum_{e,p} |E|^{rank(a_{e,p})} * |P| + \sum -p$$

Premièrement, nous avons une somme : le nombre d'étudiants puissance le rang de l'affectation p pour l'étudiant e. Ainsi, plus le rang est élevé, plus la valeur est grande, est donc plus le résultat est mauvais, puisque l'on cherche à minimiser !

Avec SCPSolver, nous faisons comme ceci :

```
double weights[] = new double[NameToId.size()];


for(UserResourceInstanceAllocation a:allowedAllocations)
{
    int id = NameToId.get(varPerAlloc.get(a));
    weights[id] = Math.pow(users.size(), prefsPerAllocation.get(a))+1;
}
```

FIGURE 3 – Modélisation de la somme de notre fonction objectif

Ensuite, on multiplie par le nombre de projets proposés puis on soustrait chaque projet affecté (uniquement dans le cas où il faut au moins une affectation par enseignant) :

```
if(inF.getOwnerAllocationPreferences()
        .equals(OwnerDesire.AT_LEAST_ONE_INSTANCE_PER_OWNER))
{
    for (int i = 0; i < weights.length; i++) {
        weights[i] = weights[i] * (varPerOwner.size()+1.0);
    }


    for(ResourceOwner ro:varPerOwner.keySet()) {
        int id = NameToId.get(varPerOwner.get(ro));
        weights[id] = -1.0;
    }
}
```

FIGURE 4 – Multiplication et soustraction

6.2 Instanciation de SCPSolver

Une fois la fonction objectif définie, nous pouvons instancier notre solveur. Ensuite nous déclarons nos variables comme étant des variables entières, puis ensuite que ces variables sont dans l'intervalle $[0, 1]$ (nous devons dire explicitement que ce sont des variables entières, étant donné que lorsqu'on définit une variable booléenne, c'est sur l'intervalle $[0, 1]$, valeurs réelles comprises). Ensuite, on dit que c'est un problème de minimisation.

```
LinearProgram lp = new LinearProgram(weights);
for (int i = 0; i < weights.length; i++) {
    lp.setInteger(i);
    lp.setBinary(i);
}

lp.setMinProblem(true);
```

FIGURE 5 – Instanciation de SCPSolver

6.3 Modélisation des contraintes

La modélisation des contraintes est plus complexe qu'avec Cplex. En effet, cplex permet d'utiliser les variables par leur nom, ce qu'on ne peut pas faire avec SCPSolver. Lorsque l'on définit une contrainte, on doit affecter un poids à **toutes** les variables, en mettant un poids de 0 pour les variables qui ne sont pas utilisées dans la contrainte.

Par exemple, voici les contrainte qui nous assurent que chaque étudiant n'est affecté qu'à un seul projet (contrainte d'égalité stricte) :

$$\sum_{p \in P} a_{e,p} = 1$$

```
for(User pl: inF.getAllUsers())
{
    double weights[] = new double[NameToId.size()];
    for(UserResourceInstanceAllocation al: inF.getResouceInstanceAllocationsFor(pl))
    {
        if(!validAllocations.contains(al))continue;
        int id  = NameToId.get(varPerAlloc.get(al));
        weights[id] = 1.0;
    }

    lp.addConstraint(new LinearEqualsConstraint(weights, 1.0,"EachUserIsGivenExactlyOneResource("+pl+")"));

}
```

FIGURE 6 – Chaque étudiant n'est affecté qu'à un seul projet

Comme dit précédemment, certains projets peuvent être proposés à plusieurs étudiants, plus globalement, chaque projet peut être affecté au maximum k fois. Nous devons donc définir des contraintes représentant ceci.

$$\sum_{e \in E} a_{e,p} <= k$$

```
for(ResourceInstance resource: allAdmissibleResources)
{
    double weights[] = new double[NameToId.size()];
    for(UserResourceInstanceAllocation ua: inF.getAllocationsForResource(resource)
            .stream()
            .filter(x->allAdmissibleAllocations.contains(x))
            .collect(Collectors.toSet()))
    {
        int id = NameToId.get(varPerAlloc.get(ua));
        weights[id] = 1.0;
    }

    lp.addConstraint(new LinearSmallerThanEqualsConstraint(
            weights,
            inF.getMaxNbUsersPerResource(),
            "EachResourceIsAllocatedAtMostKTimes("+resource+","+inF.getMaxNbUsersPerResource()+")"
    ));
}
```

FIGURE 7 – Un projet doit être affecté au maximum k fois

Ici, nous avons un exemple de contrainte **inférieur ou égal**

Il y a un grand nombre de contraintes, nous n'allons donc pas toutes les énumérer ici, cela serait trop répétitif.

6.4 Génération de la solution

Cplex offre la possibilité d'exporter le programme linéaire. CSPSolver propose également d'obtenir un programme linéaire, mais nous devons réaliser l'exportation nous même. Afin

d'avoir une ressemblance avec le programme original, nous en avons profité pour renommer toutes les variables de x_0 à x_n par leur nom que nous leur avons attribué, puis nous l'enregistrons ensuite dans un fichier.

```
String program = lp.convertToCPLEX().toString();

Map<String, Integer> sortedMap = sortByValue(NameToId);

for (Map.Entry<String, Integer> entry : sortedMap.entrySet()) {
    String varName = "_" + entry.getKey();
    int varIndex = entry.getValue();
    String plVarName = "x" + varIndex;
    program = program.replace(plVarName, varName);
}


try {
    FileWriter file = new FileWriter("output_scpsolver.lp");
    file.write(program);
    file.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

FIGURE 8 – Exportation du programme linéaire

Finalement nous pouvons générer la solution, puis traiter le résultat obtenu :

```
LinearProgramSolver solver  = SolverFactory.newDefault();
double[] sol = solver.solve(lp);

Set<UserResourceInstanceAllocation> s = processPLResults(sol, allocToVar);
```

FIGURE 9 – Génération de la solution

```
static Set<UserResourceInstanceAllocation> processPLResults(
        double[] sol,
        Map<UserResourceInstanceAllocation, String> varPerAlloc) {
    return varPerAlloc.keySet().stream()
    .filter(x->{
            return sol[NameToId.get(varPerAlloc.get(x))]>0.01;
    })
    .collect(Collectors.toSet());
}
```

FIGURE 10 – Traitement du résultat

Et notre travail s'achève donc ici.

7 Architecture

Nous avons repris l'architecture du projet de base. Nous avons commencé une implémentation du pattern Strategy afin de rendre le code plus propre et plus modulable, mais de

grandes améliorations restent à faire à ce niveau là. Nous avions une classe Solver que nous avons renommée en CplexSolver, nous avons créé une classe SCPSolver puis nous avons créé une interface Solver. Les deux classes implémentent l'interface et l'interface devrait contenir les méthodes génériques de tous les solveurs, mais ce travail n'a pas été terminé.

8 Application

Le programme final est une simple archive JAR qui peut être téléchargée accompagnée de fichiers d'exemples et de scripts d'utilisation qui permettent d'utiliser et de paramétrer facilement l'application.

Nous avons réalisé un script pour Windows et un script pour les systèmes Unix.

9 Validation

Notre tuteur de projet, nous avait proposé de réaliser une journée de validation de nos prototypes par nos pairs mais malheureusement lié aux évènements actuels la journée n'a pas eu lieu. Alors nous avons décidé de le faire valider par nos camarades. Ainsi nous avons eu des retours sur le travail fourni et à partir de ça, nous avons pu corriger nos erreurs.

Vous pouvez avoir accès au programme final à cette adresse : `https://github.com/agandois/ResourceAllocator/releases`.

Pour l'exemple fourni, nous obtenons les résultats suivants :

Level of satisfaction	Number of individuals
0	39
1	21

FIGURE 11 – Satisfaction des étudiants avec SCPSolver

Level of satisfaction	Number of individuals
0	39
1	21

FIGURE 12 – Satisfaction des étudiants avec Cplex

Number of allocated instances	Number of owners
0	9
1	1
2	8
3	1
4	4
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	2

FIGURE 13 – Satisfaction des enseignants avec SCPSolver

Number of allocated instances	Number of owners
0	9
1	0
2	9
3	0
4	5
5	0
6	0
7	0
8	0
9	0
10	1
11	0
12	1

FIGURE 14 – Satisfaction des enseignants avec Cplex

Comme nous pouvons le voir, dans les deux cas, l'insatisfaction des étudiants est la même : 39 étudiants ont une insatisfaction de 0, et 21 étudiants ont une insatisfaction de 1.

Pour la satisfaction des enseignants, il y a quelques différences, qui sont justifiées par le fait que notre objectif n'est pas nécessairement de contenter les enseignants, et donc il peut parfois y avoir de légères différences d'une implémentation à l'autre.

10 Conclusion

Dans ce projet, nous nous sommes fixé comme objectif de rendre l'application facile à installer et facile à utiliser tout en gardant des résultats similaires à l'implémentation originale et dans un temps de calcul similaire. Nous voulions aussi que notre application soit portable.

Pour ce faire, nous avons implémenté la bibliothèque CSPSolver à notre application en complément de la bibliothèque d'origine.

Nous avons montré que notre solution fonctionne en montrant sur des données d'exemple que les résultats obtenus sont similaires pour les deux implémentations.

Bien que l'application fonctionne, il reste des améliorations à faire, notamment au niveau de l'architecture logicielle. Nous pouvons également implémenter notre application au sein d'une interface web, ce qui permettrait une utilisation instantanée et de modifier les ressources en direct.