Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2022-2023

Optimisation pour la sélection de programmes TV

Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer

Entreprise
Polytech

Tuteur entreprise
Mathieu Delalandre

Étudiant
Corentin Lezé–Robert (DI5)

Tuteur académique
Tifenn Rault

4 avril 2023

Liste des intervenants

Entreprise

Polytech
64 avenue Jean Portalis
37200 Tours, France
polytech.univ-tours.fr

Nom	Email	Qualité
Corentin Lezé–Robert	email@univ-tours.fr	Étudiant DI5
Tifenn Rault	email@univ-tours.fr	Tuteur académique,Département Informatique
Mathieu Delalandre	email@univ-tours.fr	Tuteur entreprise

Avertissement

Ce document a été rédigé par Corentin Lezé–Robert susnommé l'auteur.

L'entreprise Polytech est représentée par Mathieu Delalandre susnommé le tuteur entreprise.

L'Ecole Polytechnique de l'Université de Tours est représentée par Tifenn Rault susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assume l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Corentin Lezé–Robert, *Optimisation pour la sélection de programmes TV: Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer*, Projet Recherche & Développement, Ecole Polytechnique de l'Université de Tours, Tours, France, 2022-2023.

```
@mastersthesis{
    author={Lezé–Robert, Corentin},
    title={Optimisation pour la sélection de programmes TV: Algorithmes d'optimisation
       pour la sélection de programmes TV à enregistrer},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université de Tours},
    address={Tours, France},
    year={2022-2023}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V2.5.2(2019/08/07)

Table des matières

Liste des intervenants a

Avertissement b

Pour citer ce document c

Table des matières i

Table des figures v

1 Introduction 1

1 Acteurs, enjeux et contexte 1

2 Objectifs 1

2 Description générale 2

1 Environnement du projet 2

2 Caractéristiques des utilisateurs 2

3 Fonctionnalités du système 2

4 Structure générale du système 3

3 État de l'art / Veille technologique 5

1 Article de Thomas Erlebach et Frits C.R. Spieksma 5

1.1 Complexité de l'algorithme 5

1.2 Qualité de la solution 5

1.3 $GREEDY\alpha$ 7

2 Les surveys 10

2.1 Autres variations du problème 10

2.2 Autres applications 10

4 Description des interfaces externes **11**

1 Interfaces matériel/logiciel 11

2 Interfaces homme/machine 11

3 Interfaces logiciel/logiciel 11

5 Fonctionnalités de l'algorithme **13**

1 Parsing de la journée 13

1.1 Entrée : Lien base XMLTV 13

1.2 Connexion à la base XMLTV 13

1.3 Le format XMLTV 13

1.4 Extraction des informations 16

1.5 Parseur C++ 17

1.6 Sortie : tableau en C++ 17

1.7 Tests 17

2 Calcul de la solution 18

2.1 Entrée 18

2.2 Itérations sur la solution S 18

2.3 Calcul de la sous-liste d'intervalles en conflit Qi et de i' 19

2.4 Calcul des points P_h 19

2.5 Liste des points et fonctions first() et last() 19

2.6 Calcul de coûts $C(h)$ 19

2.7 Calcul de la sous-liste d'intervalle à coût minimum Ci 20

2.8 Modification de la solution S 20

2.9 Sortie 20

2.10 Tests 20

3 Instanciation de la journée 20

3.1 Entrée 20

3.2 Filtrage des données et création du fichier CSV 20

3.3 Sortie 21

3.4 Tests 21

6 Spécifications non fonctionnelles **22**

1 Contraintes de développement et conception 22

1.1 Algorithme 22

1.2 Langage 22

1.3 Parseur 22

1.4 Interchangeabilité 22

2 Contraintes de fonctionnement et d'exploitation 23

2.1 Performances 23

2.2 Capacités 23

2.3 Modes de fonctionnement 23
2.4 Contrôlabilité 23
2.5 Donnés de test 23
2.6 Sécurité 24
2.7 Intégrité 24
3 Hypothèses 24
4 Maintenance et évolution du système 25
4.1 Seconde partie de l'algorithme GREEDY 25
4.2 Migration PC DELL 25
4.3 Boucle automatique 25
4.4 Édition des intervalles 25
4.5 Les poids 25
4.6 interface 26

7 Planification **27**
1 Premier diagramme de Gantt 27
2 Diagramme de Gantt complet 27
3 Déroulement réel 28

8 Développement **29**
1 Outils utilisés 29
1.1 Outils de code 29
1.2 Documentation 29
1.3 Organisation 29
2 Changements de structure 30
2.1 Diagramme useCase final 30
2.2 Diagramme des classes final 30
3 Changements d'organisation 31
4 Temps d'exécution 32

9 Bilan et conclusion **33**
1 Bilan du semestre 9 33
2 Bilan du semestre 10 33
3 Bilan sur la qualité 33
4 Bilan auto-critique 34

Annexes **35**

A Cahier de test **36**
1 Tests simples 36

B Documentation **37**

1 Documents disponibles 37

2 README 37

Table des figures

2 Description générale

2.1 Diagramme useCase 3

2.2 Diagramme des classes 4

3 État de l'art / Veille technologique

3.1 Schéma des sous-ensembles 8

3.2 Schéma des points 8

3.3 Schéma first et last 9

5 Fonctionnalités de l'algorithme

5.1 Diagramme du format XMLTV 15

7 Planification

7.1 Premier Gantt 27

7.2 Gantt Complet 28

8 Développement

8.1 Diagramme useCase Final 30

8.2 Diagramme des classes final 31

8.3 Diagramme de Gantt mis à jour 32

A Cahier de test

A.1 Cahier de tests 36

1 Introduction

Ce document constitue la description générale, l'état de l'art, les spécifications ainsi que le développement du projet root 5 : "Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer".

1 Acteurs, enjeux et contexte

Une station d'enregistrement est mise en place pour stocker le contenu diffusé sur les chaînes françaises. L'objectif est de stocker les diffusions afin d'y effectuer divers traitements (du fact checking pendant les campagnes présidentielles par exemple).

Cette station est gérée par M. Delalandre et plusieurs projets en découlent. Mon projet est encadré par Mme Rault.

Dans notre cas, nous nous intéressons à la problématique suivante : La station TV n'a que 8 cartes d'acquisition et ne peut donc, par conséquent, enregistrer que 8 diffusions simultanées. La solution proposée est de mettre en place un algorithme qui va, à chaque début de journée ou à chaque semaine et à partir des données du programme TV de la journée, fournir une liste de diffusions à enregistrer à la station TV.

L'algorithme que nous allons implémenter est l'algorithme $GREEDY\alpha$

2 Objectifs

L'objectif de ce projet est d'adapter et d'implémenter l'algorithme $GREEDY\alpha$ pour résoudre le problème de choix des diffusions de la station TV.

La difficulté principale sera de respecter parfaitement l'algorithme proposé afin d'obtenir un code avec une même complexité que ce qui est détaillé dans l'article sur $GREEDY\alpha$ fourni par mon encadrante. Cela dit, il faudra également veiller à ce que l'on obtienne des solutions d'une qualité suffisante. Nous aurons un indice de qualité (que nous verrons dans l'état de l'art) que le programme devra atteindre.

Le programme implémentant l'algorithme $GREEDY\alpha$ devra faire le lien entre les bases de programme TV en ligne et la station TV en fournissant une solution, c'est-à-dire une liste de programme à enregistrer, exploitable avec la station.

2 Description générale

1 Environnement du projet

Dans le cadre de mon PRD, l'environnement est libre, il n'y a pas de langage imposé, ni de milieu dans lequel l'algorithme devra tourner. L'algorithme tournera donc sur ma machine personnelle pendant tout le projet, mais il n'est pas tout à fait impossible que des tests soient effectués sur la machine DELL de la station TV.

Nous avons également fait le choix d'implémenter l'algorithme en C++ afin de rester cohérent avec les autres projets qui tournent autour de la station TV.

2 Caractéristiques des utilisateurs

L'algorithme sera manipulé par l'équipe de recherche qui gère la station TV, cela implique que :

- L'utilisateur s'y connait très bien en informatique.
  - Il maîtrise le C++.
  - Est familier avec la logique de l'algorithmique de manière générale.
- Il connait la station TV.
  - Il connaît les enjeux à respecter.
  - Il connaît le matériel.
- L'utilisateur s'y connait en ordonnancement.
  - Il connaît l'algorithme $GREEDY\alpha$.

3 Fonctionnalités du système

Les fonctionnalités seront découpées en trois grandes catégories.

- Le parsing de la journée : Toute cette partie consistera à récupérer les données de programme TV et les transformer pour pouvoir les traiter avec l'algorithme.
- La plus grosse partie du programme, l'algorithme $GREEDY\alpha$: Cette partie consistera à utiliser les données parsées afin de faire tourner notre algorithme dessus pour qu'il génère une solution. Cette partie contient également la plus grande difficulté du projet, l'implémentation fidèle de l'algorithme $GREEDY\alpha$.

— Enfin, une fois la solution trouvée, le système générera un fichier contenant la liste des programmes à suivre, qui sera utilisé pour piloter la station.

4 Structure générale du système

Diagramme useCase :

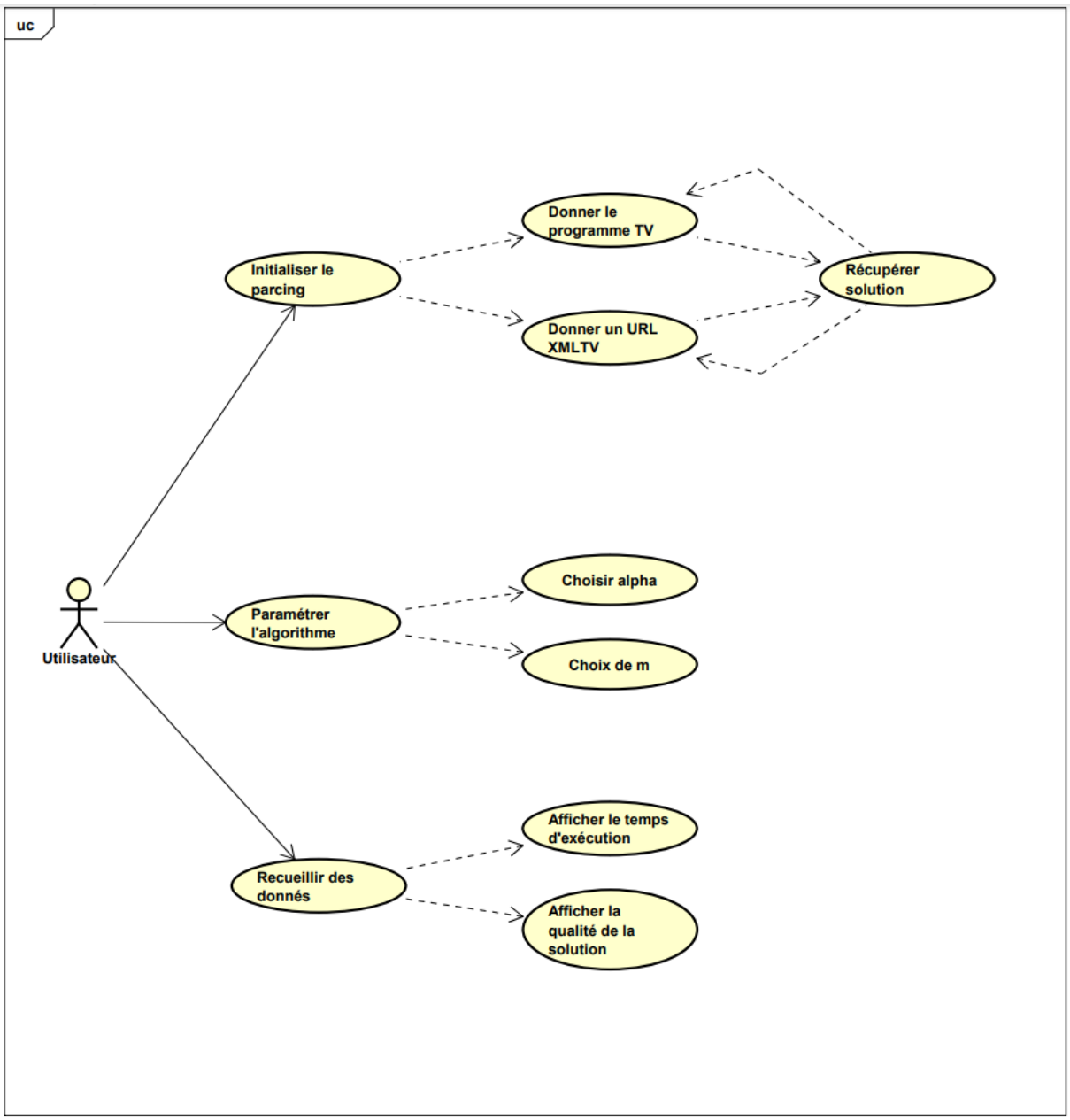


Figure 2.1 – *Diagramme useCase*

Diagramme des classes :

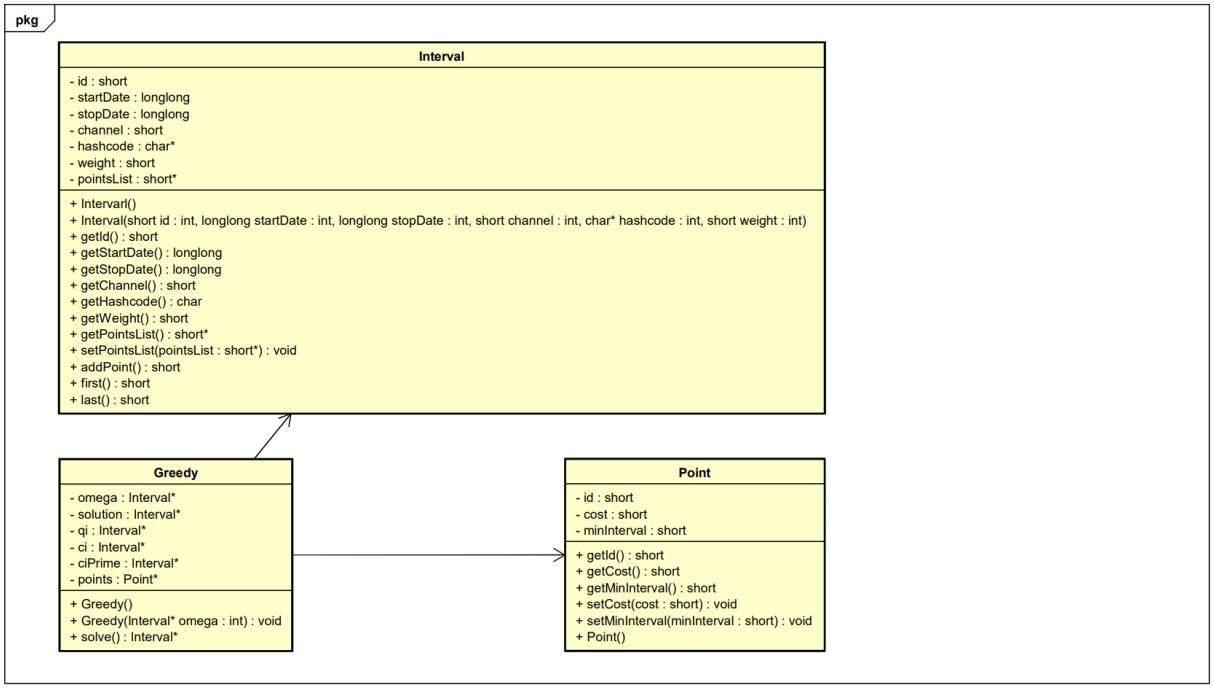


Figure 2.2 – *Diagramme des classes*

3 État de l'art / Veille technologique

Pour résoudre la problématique de ce PRD, plusieurs méthodes et algorithmes d'ordonnancement ont été étudiés, mais seul $GREEDY\,\alpha$ à été retenu.

1 Article de Thomas Erlebach et Frits C.R. Spieksma

L'article se penche de manière générale sur un problème de type "Weighted Job Interval Selection Problem" (WJISP), donc un problème de sélection d'intervalles pondérés, ce qui correspond à notre problème, car les émissions à enregistrer sont des intervalles, et des poids sont attitrés en fonction de leur importance. L'article se penche également en parallèle sur le problème WJISPm qui prend également en compte un paramètre "m" qui décrit le nombre d'intervalles simultanés que l'on tolère dans notre solution. Nous nous intéresserons plus tôt à WJISPm, car nous avons plusieurs cartes d'acquisition dans la station TV, donc plusieurs enregistrements simultanés, donc plusieurs intervalles simultanés.

1.1 Complexité de l'algorithme

Pour résoudre WJISPm, il est mentionné plusieurs algorithmes possibles, mais l'article ne décrira qu'un algorithme à passage unique, c'est-à-dire un algorithme qui ne parcourt qu'une seule fois la liste d'intervalles et détermine, pour chaque intervalle, s'il est ajouté à la solution ou définitivement rejeté. Cela nous intéresse, car pour notre problème d'ordonnancement, nous souhaiterions implémenter un programme qui s'exécute très vite et le fait que l'algorithme soit à passage unique permet d'avoir une complexité très basse. Le code n'itérera pas plusieurs fois sur la liste des intervalles, contrairement à la grande majorité des algorithmes proposés et cela va nous permettre de gagner du temps d'exécution.

1.2 Qualité de la solution

L'article fait la distinction entre plusieurs variantes plus ou moins compliqués de WJISP :

— JISP : Une version du problème d'ordonnancement d'intervalles où les intervalles n'ont pas de poids, ils ont tous la même "importance".

- WJISP :
  - Avec des poids arbitraires, mais où chaque intervalle dans un même job a forcément le même poids, sachant qu'un job est un groupe d'intervalles identiques. Pour conceptualiser, dans le cadre de notre problème, un job serait une émission et un intervalle une diffusion de cette émission. Cela veut donc dire que même si une émission est diffusée plusieurs fois, on souhaite qu'à chaque fois, elle ait le même poids.
  - Ou bien avec des poids complètement arbitraires, même dans un même job.
- WJISPm : Enfin, WJISP avec "m" intervalles parallèles autorisés (existe également en version poids uniforme dans les jobs ou non).

Pour estimer l'efficacité de l'algorithme, on estime la qualité d'une solution dans le pire cas "ρ", un ratio qui exprime par combien la qualité de la solution a été divisée par rapport à une solution optimale. Par exemple, si ρ vaut 1, la solution est optimale, si ρ vaut 4, la solution sera dans le pire des cas 4 fois moins bonne que la solution optimale.

Il existe déjà des recherches faites sur certaines variantes de WJISP ou sur d'autres problèmes similaires, certaines ainsi que leur valeur ρ déterminée sont détaillées dans l'article :

- JISP : Spieksma estime ρ environ égale à 2 si l'on applique un simple algorithme greedy au problème.
- TCSP sans poids : Un problème qui peut être considéré comme un cas spécifique de WJISP. Dans ce problème, il y a des tâches qui ont une date d'émission de la tâche, une date butoir, une durée ainsi qu'un poids (pour comprendre la différence avec WJISP, on peut imaginer une tâche qui a 4h entre l'émission et la date butoir, mais qui ne met que 1h à compléter, cela donne donc une zone dans laquelle les intervalles peuvent être bougés). Dans le cas de TCSPm avec les poids tous fixés à 1, Bar-Noy et al. trouvent un ρ qui dépend de m avec les valeurs suivantes : $\rho(1) = 2$ et $\rho(m) \approx 1.582$ si m tend vers l'infini. Ces résultats sont obtenus en appliquant plusieurs itérations de GREEDY.
- TCSP poids égaux par job : Bar-Noy et al. obtiennent $\rho(m) \approx 3+2\sqrt{2} \approx 5.828$ en appliquant ADMISSION qui est un algorithme qui applique une procédure greedy n fois sur la liste d'intervalles.

ADMISSION pourrait tout à fait être appliqué à notre problème WJISPm, en revanche comme explique précédemment ADMISSION est un algorithme qui itère plusieurs fois, nous voulons garder le temps d'exécution le plus court possible, nous cherchons donc à implémenter un algorithme le moins complexe possible (c'est également ce sur quoi se penche l'article, un algorithme à passage unique). Cela dit, les valeurs ρ nous permettent d'estimer le genre de résultat auquel nous pourrions nous attendre pour $GREEDY\alpha$, avec les valeurs exprimées plus tôt qui seront des limites vers lesquels $GREEDY\alpha$ pourra tendre.

L'article décrit ensuite les résultats obtenus avec $GREEDY\alpha$ dans plusieurs variantes de WJISP, il y a également une distinction entre un cas avec des intervalles de tailles égales et des intervalles de tailles aléatoires, voici les résultats :

Intervalles à tailles égales		
Variante de WJISP	Ratio de $GREEDY\,\alpha$	Meilleur cas
JISP	$2, \forall m \geq 1$	$2, \forall m \geq 1$
WJISP poids égaux par job	$5, m = 1, 2$	$5, m = 1$
WJISP poids arbitraires	$\approx 6.638 m = 1, 2$	$3 + 2\sqrt{2} \approx 5.828, \forall m \geq 1$
Intervalles à tailles arbitraires		
Variante de WJISP	Ratio de $GREEDY\,\alpha$	Meilleur cas
JISP	$2, \forall m \geq 1$	$2, \forall m \geq 1$
WJISP poids égaux par job	$3 + 2\sqrt{2} \approx 5.828, \forall m \geq 1$	$3 + 2\sqrt{2} \approx 5.828, \forall m \geq 1$
WJISP poids arbitraires	$8, \forall m \geq 1$	$\approx 7.103, \forall m \geq 1$

1.3 $GREEDY\,\alpha$

L'algorithme $GREEDY\,\alpha$ est décrit en plusieurs étapes, dans un premier temps, je vais expliquer ce que fait l'algorithme en parcourant la liste des intervalles sans se soucier des détails, ou comment certaines choses sont trouvées. Nous irons ensuite voir les sous fonctions plus en détail.

Cela n'est pas spécifié dans l'article, mais pour l'explication, j'appelle la liste des intervalles Ω.C'est une suite d'intervalles avec les propriétés suivantes :

- Une heure de début
- Une heure de fin
- Un poids

Cette liste va être parcourue une seule fois par $GREEDY\,\alpha$, nous appellerons l'intervalle d'Ω en cours de traitement i. L'algorithme va pour chaque i essayer de l'inclure dans une autre liste d'intervalle S qui sera notre solution. Pour savoir si i peut être ajouté à la solution, on le compare à Ci qui est le sous ensemble de S à coût minimum que l'on doit retirer pour faire rentrer i. Ici la difficulté est de trouver Ci, c'est ce que nous allons voir après. À noter que c'est ici que le paramètre α rentre en compte, effectivement, pour faire le choix entre Ci et i on compare leur poids total. Dans $GREEDY$ normale, on garde tout simplement l'intervalle ou le sous ensemble d'intervalle qui a le plus haut poids, mais dans $GREEDY\,\alpha$ on multiplie le poids de i par α, permettant de modifier S même si l'échange entre Ci et i n'augmente pas toujours le poids total de la solution.

Voici à quoi ressemble l'algorithme pour le moment (Sachant que l'on note le poids d'un intervalle ou sous ensemble $\omega(intervalle)$) :

$S \leftarrow \emptyset$
for $\forall i \text{dans} \Omega \text{dans l'ordre croissant des dates de début}$ **do**
  $Ci \leftarrow$ Sous ensemble à coût minimal tel que $(S \setminus Ci) \cup i$ soit faisable
  if $\omega(Ci) \leqslant \alpha\omega(i)$ **then**
    $S \leftarrow (S \setminus Ci) \cup i$
  end if
end for
return S

L'article explique ensuite comment trouver Ci. Dans le cas où notre nombre d'intervalles parallèles tolérés m est égal à 1, Ci est assez simple à trouver, car il contient juste la liste de tous les intervalles de S qui entrent en conflit avec i. Vu que $m = 1$ nous n'avons pas à nous soucier des poids et les intervalles en conflit sont ceux qui se déroulent en partie ou totalement en même

temps que i ou bien un intervalle i' qui serait une précédente diffusion de l'émission i (ici i et i' font donc partie du même job et on ne souhaite qu'un seul intervalle par job).

Maintenant dans le cas où $m > 1$, le problème est un peu plus compliqué, car parmi tous les intervalles qui coupent i, seuls certains doivent être retirés pour laisser rentrer i. On appelle donc Qi le sous ensemble de S contenant tous les intervalles qui coupent i, à bien distinguer de Ci qui est un sous ensemble à coût minimal de Qi (avec i' en plus). Voici un schéma pour visualiser la situation.

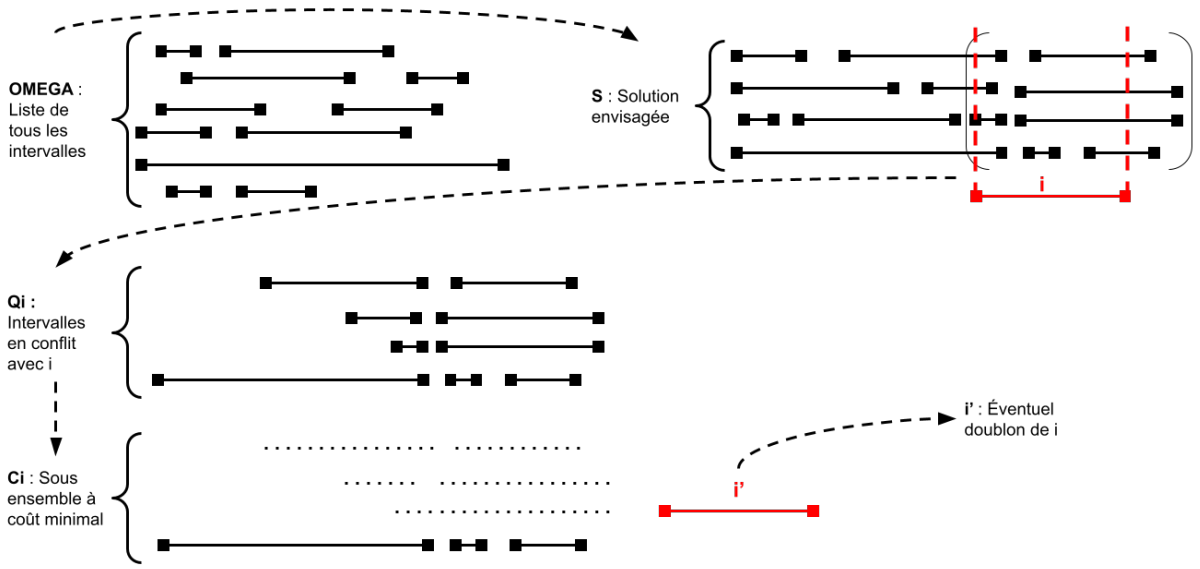


Figure 3.1 – *Schéma des sous-ensembles*

On note également Ci' l'intervalle à coût minimum ne contenant pas i'. C'est Ci' qui sera compliqué à trouver, car s'il y a un i' dans S, il sera forcément présent dans Ci, il n'y a pas de choix à faire à ce niveau-là.

L'article décrit en suite que Ci' peut être trouvé à partir de Qi en un temps $O(n)$ en utilisant une méthode décrite plus tard dans l'article, mais pour le moment, nous nous concentrons sur une méthode en $O(nm)$ avec la méthode suivante. Pour parcourir Qi, on appelle les intervalles contenus dedans $j_l \in Qi$, avec $1 \leqslant l \leqslant s$. On attribue des points p_h avec $1 \leqslant h \leqslant r$ juste avant la fin de chaque intervalle j_l, s'il coupe m intervalles au total, alors au moins l'un de ces intervalles doit être contenu dans Ci'. On garde l'ensemble de points $p_1, ..., p_r$ tel que chaque point p_l coupe m intervalles. Voici un nouveau schéma pour représenter la situation.

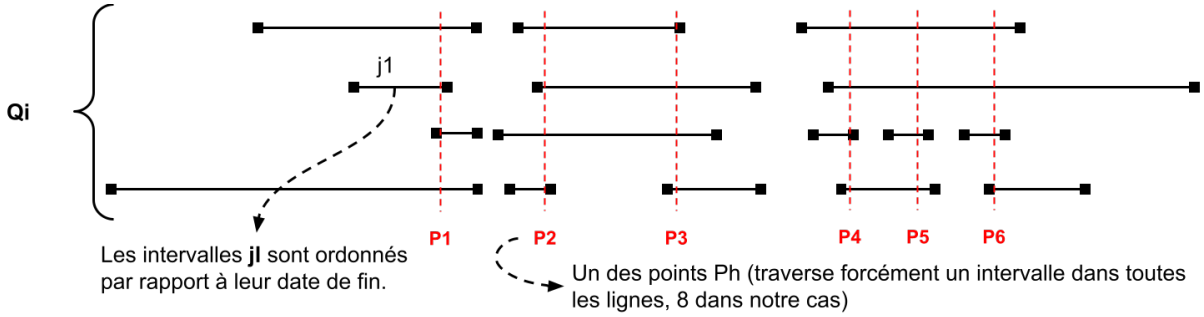


Figure 3.2 – *Schéma des points*

On donne en suite $first(j_l)$ et $last(j_l)$ qui sont respectivement les premiers et derniers p_h qu'un intervalle j_l coupe, si j_l ne coupe pas de p_h, $first(j_l)$ et $last(j_l)$ ne sont pas définis. Voici une représentation visuelle :

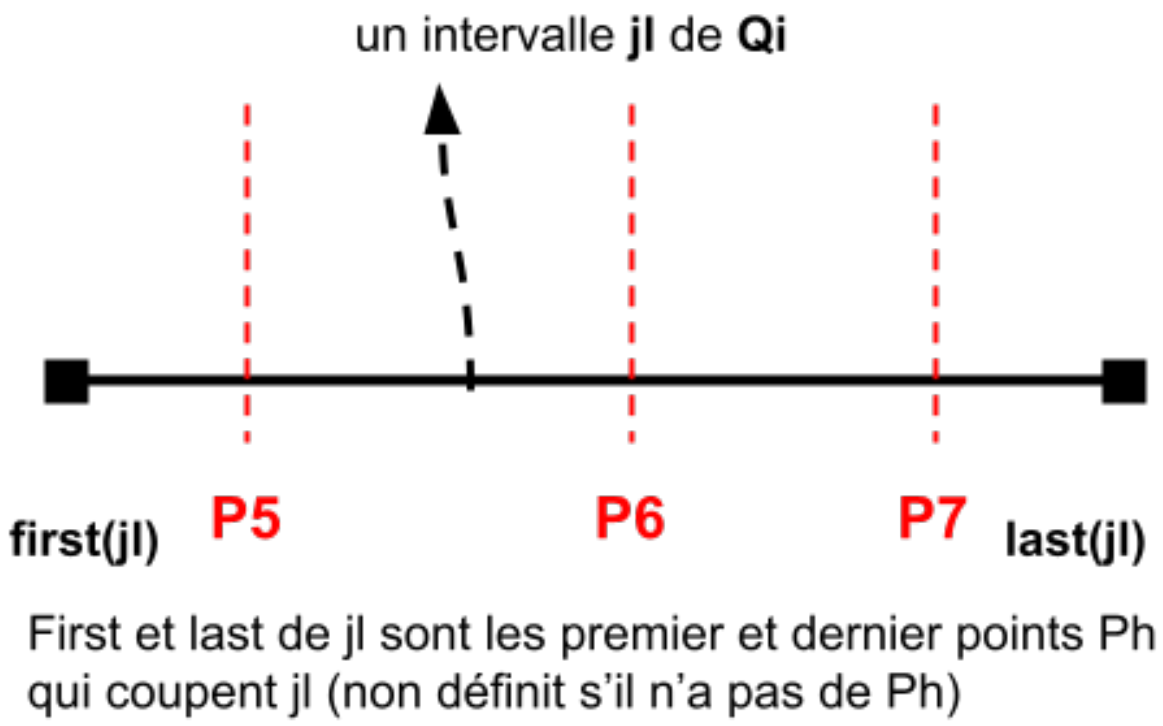


Figure 3.3 – *Schéma first et last*

On remplit un tableau contenant tous les $first(j_l)$ et $last(j_l)$. Les intervalles j_l, les points p_h, les $first(j_l)$ et les $last(j_l)$ peuvent tous êtres construits en un temps $O(nm)$. En gardant la même approche et en restant dans la même boucle pour ne pas augmenter la complexité, on calcule des coûts à chaque point, notés $C(h)$, toujours avec $1 \leqslant h \leqslant r$. On attribue les coûts $C(h)$ de manière à ce qu'après avoir itéré jusqu'au $l^{ième}$ intervalle, le coût $C(h)$ soit équivalent au poids du sous-ensemble à coût minimal $\omega(j_1,...,j_l)$ qui couvre les points $p_1,...,p_h$. Les $C(h)$ sont initialisés à ∞, ce qui veut dire que si $first(j_l)$ et $last(j_l)$ ne sont pas définis, on gardera un $C(h) = \infty$. Les coûts $C(h)$ sont donc de plus en plus hauts au fur et à mesure que l'on avance dans les points. Après avoir itéré sur tous les intervalles, on obtient le coût du dernier point $C(r) = \omega(Ci')$, le sous ensemble à coût minimal que l'on recherche. Voici l'algorithme à appliquer :

$C(0) \leftarrow 0$
for $h = 1$ à $h = r$ **do**
    $C(h) \leftarrow \infty$
end for
for $l = 1$ à $l = s$ **do**
    if $first(j_l)$ n'est pas indéfini **then**
        for $h = first(j_l)$ à $h = last(j_l)$ **do**
            $C(h) \leftarrow min(C(h), C(first(j_l) - 1) + \omega(j_l))$
        end for
    end if
end for

Cela n'est pas spécifié dans l'article, mais cet algorithme ne nous donne que le poids du sous ensemble minimal pour le moment, il faut également garder en mémoire le j_l qui donne à un point son coût minimal lors du calcul de $C(h) \leftarrow min(C(h), C(first(j_l) - 1) + \omega(j_l))$. Le sous ensemble Ci' sera donc la liste des j_l attitré à chaque point.

L'article explique dans un second temps une suite à cette solution permettant d'effectuer

$GREEDY\alpha$ en $O(n)$, mais nous n'implémenterons pas cette solution dans ce projet et nous nous contenterons d'un algorithme en $O(nm)$.

2 Les surveys

Nous analysons également deux articles Survey sur les problèmes de prises de décision d'intervalles. Ces articles nous permettent d'avoir une vision plus large sur le problème, ses applications, ses variantes, etc.

Ces articles sont : "Fixed interval scheduling : Models, applications,computational complexity and algorithms" de Mikhail Y. Kovalyov , C.T. Ng, T.C. Edwin Cheng et "Interval Scheduling : A Survey" de Antoon W.J. Kolen, Jan Karel Lenstra, Christos H. Papadimitriou, Frits C.R. Spieksma

2.1 Autres variations du problème

Il est intéressant d'explorer un peu davantage de variations de notre problème, cela peut permettre de mieux comprendre comment évolue la complexité et cela permet aussi de mieux comprendre où et comment s'inscrit notre problème dans son domaine. Voici les variantes évoquées dans les Surveys :

- Disponibilité des machines : On considère que durant certains intervalles de temps, le nombre d'intervalles simultanés m diminue, car certaines machines ne sont pas disponibles.
- Une variante dans laquelle les intervalles se superposent par pair.
- Un problème différent ou chaque job doit forcément avoir un de ses intervalles pris dans la solution.
- Un problème différent où les intervalles possèdent de possibilités discrètes de point de départ.
- Un problème dans lequel certains jobs ne peuvent être réalisés que sur certaines machines.

2.2 Autres applications

Des domaines d'applications sont également mentionnés dans les articles, ceux-ci peuvent fournir des pistes de recherche intéressantes pour faire évoluer le projet, voici les domaines cités :

- Gestion de flottes.
- Câblage d'ordinateur.
- Allocation de bande passante.
- Gestion des transports (Bus / train / Avion).
- Gestion d'emploi du temps (Scolaire par exemple).
- Gestion du personnel.
- Impression de carte électronique.
- Biologie moléculaire (Comparaison de génome).
- Spectroscopie.
- Problèmes de disparités entre le processeur et la vitesse mémoire d'un ordinateur.
- Gestion de photographie satellite.
- Réservation client.
- Allocation de mémoire.

On remarque que les domaines sont très variés voir très éloignés, il peut donc y résider des solutions utiles, mais complètement inconnu dans notre milieu à cause de cet éloignement.

4 Description des interfaces externes

1 Interfaces matériel/logiciel

Le contrôle de la station en elle-même n'est pas de mon ressort dans ce projet, cette tâche est donnée à un autre groupe d'étudiants en 4A. En revanche, le contrôle de la station sera effectué en fonction du résultat que l'algorithme GREEDY retourne. Ce résultat prend la forme d'un fichier CSV formaté correctement. Même s'il n'y a pas d'interface matériel/logiciel dans ce projet, la station sera indirectement contrôlée par le fichier CSV, ce qui sera à prendre en compte (surtout pour de légères raisons de sécurités que nous définissons plus tard).

Dans le cadre de ce projet, j'enverrai les fichiers CSV manuellement au groupe de 4A, ce qui ne génère donc aucune interface.

2 Interfaces homme/machine

L'interface homme/machine sera assez simple, l'utilisateur pourra manipuler le programme en console de commandes proposant les options suivantes :

- La possibilité de fournir les données d'un programme TV.
- Le choix des deux valeurs "alpha" et "m" qui sont des paramètres de $GREEDY\alpha$.
- Afficher les statistiques calculées.

Cette interface sera suffisante, au vu du fait que je serai le seul à manipuler l'algorithme, sur ma machine personnelle. Même si l'équipe de la station TV était amenée à l'utiliser, l'utilisation d'une console de commandes ne causera aucun souci.

3 Interfaces logiciel/logiciel

Comme précisé précédemment, l'algorithme devra pouvoir prendre la liste des programmes d'une journée en entrée, ces programmes sont regroupés dans un fichier XMLTV qui est le format utilisé par les services qui mettent à disposition les programmes TV en ligne. Cela génère une première interface logiciel/logiciel, car nous voulons récupérer automatiquement ces fichiers en établissant une connexion avec la base en ligne.

Il y aura une seconde interface entre mon programme et un programme de parsing déjà existant. Effectivement un programme python faisant déjà une grande partie du parsing est fourni, je vais donc le réutiliser, l'adapter et lui ajouter la possibilité de se connecter à la base en ligne (car il nécessite de lui fournir un fichier XMLTV manuellement pour le moment). Ce script python fera donc la liaison entre la base en ligne et le reste du programme en C++.

5 Fonctionnalités de l'algorithme

1 Parsing de la journée

Cette section concernera principalement le programme python fourni.

Cela dit, une partie du parsing sera également faite en C++. Effectivement, étant donné que plusieurs projets peuvent exploiter le parsing fait par le code python, nous souhaitons garder son parsing tel quel (bien qu'en améliorant un peu l'algorithme), puis refaire un second passage en C++ afin de peaufiner le parsing pour l'algorithme. Le code python ne fera qu'un fichier CSV appelé fichier collection, que le code C++ va récupérer et re-parser.

1.1 Entrée : Lien base XMLTV

L'algorithme prendra en entrée une URL permettant de se connecter à la base de données de programme TV souhaitée.

1.2 Connexion à la base XMLTV

Dans un premier temps, l'algorithme va récupérer une liste de diffusion pour la journée. Pour cela, le programme devra se connecter à une base de données STVD-PMS, les intervalles sont récupérés au format XMLTV, où toutes les diffusions seront stockées dans une même liste.

1.3 Le format XMLTV

Ce format est un format XML classique, il est donc composé de balises hiérarchisées qui contiennent des informations. Les balises peuvent être ouvertes et fermées, ce qui veut dire que des informations peuvent être contenues dans l'entête de la balise ou dans son corps (entre son ouverture et sa fermeture). D'autres sous-balises sont ouvertes dans le corps de la balise principale, c'est ainsi que la hiérarchie se forme.

Le fichier prend la forme suivante : Tout est contenu dans une grande balise TV. Dans cette balise TV sont d'abord listées les balises chaînes, puis les balises programmes. D'autres balises

sont présentes dans les balises chaînes et programmes, mais celles-ci n'ont pas d'ouvertures ou fermetures, elles contiennent juste une information dans leurs entêtes.

La balise TV contient les informations suivantes dans son entête :

— Une URL de source d'informations.
— Une URL de source de donnés (Les informations et les données proviennent généralement de la même source).
— Une URL du générateur du fichier XMLTV.
— Le nom du générateur (ici XMLTV).

Puis les balises suivantes dans son corps :

— Les balises chaînes.
— Les balises programmes.

Les balises chaînes contiennent les informations suivantes :

— L'identifiant de la chaîne, dépends de la source d'informations/donnés utilisée.

Puis les balises suivantes dans leurs corps :

— Le nom de la chaîne à afficher.
— Une URL vers une image servant d'icône à la chaîne.

Les balises programmes contiennent les informations suivantes :

— Une heure de début.
— Une heure de fin.
— L'Id de la chaîne sur laquelle le programme est diffusé.

Puis les balises suivantes dans leurs corps :

— Le titre du programme diffusé.
— Un sous-titre (optionnel).
— Une description brève (optionnel).
— Une catégorie de programme (certains programmes sont non déterminés).
— La durée du programme (en minutes).
— Une URL vers une image servant d'icône à l'émission.
— Une balise du type nouveau / déjà diffusé / première (ici ce n'est pas une balise qui contient une information, mais le type de la balise elle-même qui sert d'indicateur).
— Un numéro d'épisode au format suivant : saison.épisode (optionnel).
— Une note sur 5, corresponds aux notes de 5 étoiles (optionnel).
— Une balise signalétique qui prend le type de signalétique en paramètre et une balise note en son corps.
— Une balise audio qui n'a pas de paramètres, mais possède une balise qui correspond au type d'audio (stéréo / audio ...) qui contient la langue en son corps.

Voici un schéma pour mieux représenter la hiérarchie.

- **Balise TV**
  - URL info | URL donnés | URL générateur | Nom générateur
    - **Balise Chaine**
      - Id chaine
        - Balise nom
        - Balise icône
    - **Balise Programme**
      - heure de début | heure de fin | Id chaine
        - Balise titre
        - Balise sous-titre
        - Balise description
        - Balise catégorie
        - Balise durée (minutes)
        - Balise icône
        - Balise déjà diffuse / nouveau / première
        - Balise date
        - Balise numéro d'épisode
        - Balise note étoiles
        - **Balise Signalétique**
          - Type de signalétique
            - Balise note
        - **Balise Audio**

Figure 5.1 – *Diagramme du format XMLTV*

Voici un exemple avec une chaîne et un programme :

```
<tv source-info-url="https://api.telerama.fr" source-data-url="https://api.telerama.fr" generator-info-name="XMLTV" generator-info-url="http://mythtv-fr.org/">
  <channel id="C192.api.telerama.fr">
    <display-name>TF1</display-name>
    <icon src="https://television.telerama.fr/sites/tr_master/files/sheet_media/tv/500x500/192.png" />
  </channel>
  <programme start="20221123095400 +0100" stop="20221123115700 +0100" channel="C34.api.telerama.fr">
    <title>Downton Abbey II : Une nouvelle ère</title>
    <sub-title>Sous-titre</sub-title>
    <desc lang="fr">Dans le Yorkshire, en 1928. Réunis lors de la célébration d'un mariage, les Crawley apprennent qu'un notaire doit prochainement se rendre au château pour leur annoncer une grande nouvelle. En effet, Violet, la mère de Lord Grantham, découvre que le défunt marquis de Montmirail la désigne comme héritière de la villa des Colombes, une superbe demeure située dans le sud de la France. Intrigué, Lord Grantham souhaite comprendre les mystérieuses raisons d'un tel legs. Par ailleurs, face à l'insistance d'un réalisateur, il accepte de céder à sa demande et l'autorise à faire de son château un lieu de tournage pour un long métrage...</desc>
    <date>2022</date>
    <category lang="fr">film : drame</category>
    <length units="minutes">123</length>
    <icon src="https://focus.telerama.fr/1111x625/2021/12/22/9bf25a48-106e-5b16-1984b3d5c1efe.jpg" />
    <audio>
      <stereo>bilingual</stereo>
    </audio>
    <previously-shown />
    <episode-num system="xmltv_ns">9.32.</episode-num> <!--numéro d'épisode factice ajouté pour la démonstration -->
    <rating system="CSA">
      <value>Tout public</value>
    </rating>
    <star-rating>
      <value>4/5</value>
    </star-rating>
  </programme>
</tv>
```

1.4 Extraction des informations

Une fois les données récupérées, le programme va filtrer et réorganiser les parties intéressantes afin de contenir tous les programmes dans un tableau à deux dimensions. Chaque ligne sera une

diffusion et chaque colonne une donnée sur la diffusion.

Chaque diffusion contiendra les informations suivantes :

- La chaine sur laquelle elle est diffusée
- Un hashcode qui lui sert d'identifiant. Si une émission est diffusée à plusieurs reprises, elle garde le même hashcode sur toutes ses diffusions. Ce hashcode sera créé en hachant le titre de l'émission.
- Une heure de début et une heure de fin. Elles sont notées en secondes passées depuis l'heure de début des diffusions (6h00).
- Un poids attitré aléatoirement (généré en fonction de son hashcode).
- Une latence
- Une moyenne
- Une déviation standard

À noter que les trois derniers éléments (latence, moyenne, déviation standard) ne nous seront pas du tout utile dans le cadre de ce projet. Le code python génère des fichiers que l'on appelle "collection" et qui peuvent être utiles pour d'autres projets, c'est pour ça que nous faisons un deuxième parsing en C++.

1.5 Parseur C++

Un lien entre le code Python et le code C++ sera effectué avec le fichier CSV généré par le code python.

Ce petit parseur aura pour seul objectif d'éliminer les trois dernières informations et de trier les intervalles par ordre croissant (calage à gauche).

1.6 Sortie : tableau en C++

Une fois les informations parsées, tout sera stocké dans un tableau, qui contiendra les informations suivantes :

- La chaine sur laquelle elle est diffusée.
- Un hashcode qui lui sert d'identifiant.
- Une heure de début et une heure de fin. Elles sont notées en secondes passées depuis l'heure de début des diffusions (6h00).
- Un poids.

1.7 Tests

Afin de tester le bon fonctionnement de la partie parsing, il y aura plusieurs choses à vérifier :

- La connexion à la base
- La qualité du parsing
- le bon passage en C++

Pour la connexion à la base, cela demande juste de vérifier qu'il n'y a pas d'erreurs de connexion et que le fichier est bien téléchargé.

Pour la qualité du parsing, je vais comparer une petite instance à une version déjà parsée, il faudra vérifier les critères suivants dans de tests unitaires :

- Seules les données voulues sont extraites.
- Il ne manque aucunes données.

— Le hashcode est bon (vérifier par rapport au hachage appliqué manuellement).
— Les émissions présentes plusieurs fois possèdent bien toutes le même hashcode.
— Il n'y a pas de caractère problématique.
— Il n'y a pas d'émissions superposées sur une même chaîne (ou autre aberration).

2 Calcul de la solution

Une fois le programme de la journée récupéré, nous allons pouvoir appliquer l'algorithme. Ici, une diffusion prend la forme d'un intervalle avec une heure de début, une heure de fin et un poids qui reflète l'importance de l'émission.

L'algorithme GREEDY ne passe qu'une seule fois sur la liste d'intervalles Ω à ordonner (ce qui permet une complexité faible).

2.1 Entrée

L'algorithme prendra en entrée le tableau d'intervalles Ω généré par notre parseur décrit plus tôt.

Nous avons également vu plus tôt que l'algorithme $GREEDY\,\alpha$ prend deux paramètres en entrée, α et m :

— m est le nombre d'intervalles simultanés que nous tolérons. Cette valeur reflète le fait que nous avons un nombre limité de cartes d'acquisition et donc un nombre limité d'enregistrements simultanés.
— alpha est un ratio que nous utiliserons pour comparer les poids de deux sous ensembles d'intervalles.

2.2 Itérations sur la solution S

Durant l'exécution de l'algorithme, nous allons stocker une solution S que nous allons modifier au cours du parcours de la liste des intervalles Ω. En parcourant cette liste Ω, l'algorithme va essayer d'insérer chaque intervalle de Ω dans S, il faudra donc vérifier que ce nouvel intervalle i que nous tentons d'insérer ne génère aucun conflit.

À chaque fois que l'on tente d'insérer un nouvel intervalle i, il faut vérifier les trois conditions suivantes :

— L'intervalle i se positionne-t-il à un endroit où il y aurait plus de m intervalles simultanés ? Si oui, cela implique de retirer un ou plusieurs intervalles pour faire rentrer i.
— Un intervalle du même job que i est-il déjà présent dans S ? Effectivement certains intervalles sont présents plusieurs fois dans Ω sur des horaires différents, nous pouvons voir qu'il s'agit du même job, car i et l'autre intervalle identique i' possèdent le même hashcode. Cela reflète qu'une émission peut être diffusée plusieurs fois, ici un job correspond donc à une émission, (un épisode d'une série par exemple) et tous les intervalles compris dans ce job sont ses diffusions. Comme nous ne voulons pas enregistrer plusieurs fois la même émission, si i' est déjà présent dans S, insérer i implique donc de supprimer i'.
— Si l'une des deux conditions précédentes est vérifiée, il va falloir comparer le poids de i et le poids du sous ensemble d'intervalles Ci que nous sommes sur le point de retirer pour insérer i. C'est ici que nous allons utiliser alpha, on note les poids de Ci et i , $\omega(Ci)$ et $\omega(i)$, on ne remplace Ci par i que si $\omega(Ci) \leq \alpha * \omega(i)$.

2.3 Calcul de la sous-liste d'intervalles en conflit Qi et de i'

Comme nous l'avons vu plus tôt, pour trouver le sous ensemble à coût minimal Ci', il nous faut itérer sur la liste des intervalles en conflit Qi. Pour déterminer Qi, une fonction itérera sur notre solution S et regardera tout simplement si l'intervalle en cours de traitement à une date de fin inférieure à la date de début de i où une date de début inférieure à la date de fin de i. Si l'une des deux conditions est vérifiée, l'intervalle est ajouté à Qi.

Durant notre parcours, on regarde également les hashcodes de nos intervalles afin de déterminer s'il existe un doublon de l'émission i. Nous n'ajoutons pas i' dans Qi mais nous gardons une liste des i' à part.

2.4 Calcul des points P_h

Nous allons construire les points P_h, nous allons parcourir notre liste Qi pour y détecter les points. Pour savoir si un point existe, il faut qu'un point situé juste avant la fin d'un intervalle coupe au total m intervalles. Pour trouver cela, nous allons à chaque intervalle j_l regarder tous les intervalles d'indice supérieur, si cet intervalle à une date de début inférieure à la date de fin de j_l. Si cette condition est vérifiée, on incrémente un compteur (initialisé à 1, car un point coupe toujours au moins l'intervalle d'où il est issu) et si ce compteur atteint m, alors nous avons trouvé un point P_h. On incrémente également la valeur h pour pouvoir donner des indices à chaque point.

À noter qu'à chaque fois que l'on trouve un point, on garde en mémoire une liste des intervalles qui ont été coupés, afin de pouvoir enregistrer dans ces intervalles, que le nouveau point p_h qui vient d'être créé, les coupent.

2.5 Liste des points et fonctions first() et last()

Chaque intervalle j_l possède en attribut la liste des indices des points qui le traverse, comme nous l'avons vu précédemment. Les deux fonctions seront appelées first() et last() pour le calcul des poids $C(h)$ et retournent tout simplement le plus petit et le plus grand indice de la liste des points. Contrairement à ce qui expliqué durant l'état de l'art, nous n'allons pas remplir un tableau différent, mais ce sera bien nos intervalles qui posséderont cette information.

2.6 Calcul de coûts $C(h)$

Calculer les poids reviens à une implémentation identique de l'algorithme vu durant l'état de l'art. Chaque point possède un coût et nous les initialisons tous à ∞. On parcourt ensuite tous nos intervalles j_l de Qi et pour chacun de ces intervalles, on parcourt les points p_h qui les coupent. Pour chaque point p_h, on met à jour son coût $C(h)$ selon la condition suivante :

$$C(h) \leftarrow min(C(h), C(first(j_l) - 1) + \omega(j_l))$$

On garde également en mémoire l'indice de l'intervalle qui donne le coût minimum lors de cette condition, en le stockant dans une liste de parcours. Cette liste correspond à Ci'.

2.7 Calcul de la sous-liste d'intervalle à coût minimum Ci

Avant de regarder si on insère i, il faut récupérer la liste des intervalles à retirer. Nous possédons déjà le sous ensemble à coût minimal Ci' ainsi que son poids (qui correspond au coût du dernier point P_r). Il nous reste donc à ajouter i' à i, ainsi que $\omega(i')$ à $\omega(Ci')$ pour obtenir Ci et $\omega(Ci)$.

2.8 Modification de la solution S

Nous possédons toutes les informations nécessaires, il ne reste plus qu'à vérifier la condition de remplacement de Ci par i :

$$\omega(Ci) \leqslant \alpha\omega(i)$$

Si cette condition est vérifiée, on retire Ci de S pour ajouter i, sinon on ne fait rien.

2.9 Sortie

Une fois le parcours d'Ω terminé, on obtient un tableau S avec tous les intervalles choisis. Cet algorithme renvoi ce tableau en sortie

2.10 Tests

Pour tester le bon fonctionnement de l'algorithme (à ne pas confondre avec les performances : rapidité, qualité de solution ...), je vais comparer la solution trouvée par l'algorithme avec une solution faite à la main. Il faudra que la solution de l'algorithme valide les critère suivants :

— Pas plus de m intervalles simultanés

3 Instanciation de la journée

Une fois la solution trouvée, il faut créer une liste de programmes à enregistrer exploitable par la station TV. La solution prendra la forme d'un fichier CSV dans lequel les intervalles sont mis en m listes parallèles, avec leurs heures de début et de fin.

3.1 Entrée

Ici, nous allons récupérer la solution fournit par l'algorithme, c'est-à-dire une liste C++ d'intervalles contenant les informations suivantes :

— Une date de début
— Une date de fin
— Une chaine
— Un hashcode
— Une liste de points.

3.2 Filtrage des données et création du fichier CSV

La station TV ne reconnait pas les émissions, tout ce qu'il faut pour la piloter, c'est une heure de début d'enregistrement, une heure de fin d'enregistrement et une chaîne sur laquelle enregistrer. Nous allons donc tout simplement extraire ces données et les écrire toutes à la suite dans un fichier CSV.

3.3 Sortie

L'algorithme donnera en sortie le fichier CSV créé.

3.4 Tests

Pour tester cette fonctionnalité, il faudra tout simplement créer des fichiers CSV à la main à partir de solutions de GREEDY et comparer que toutes les données sont parsées correctement.

6 Spécifications non fonctionnelles

1 Contraintes de développement et conception

1.1 Algorithme

La première contrainte à respecter est celle de l'algorithme. Il est vrai que *GREEDY α* peut être remis en question et implique des sacrifices sur la qualité de la solution, mais dans le cadre de ce projet, c'est uniquement *GREEDY α* qui à été sélectionné et il devra impérativement être implémenté.

1.2 Langage

Comme exprimé précédemment, *GREEDY α* sera implémenté en C++ par souci de cohésion avec les autres projets autours de la station TV.

1.3 Parseur

Même si *GREEDY α* sera implémenté, il faudra que la majeure partie du parsing soit fait en python, car le programme existe déjà et d'autres sujets peuvent en dépendre. Le reste du parsing spécifique à *GREEDY α* sera fait en C++.

1.4 Interchangeabilité

L'implémentation de *GREEDY α* devra être séparée des autres fonctionnalités afin de pouvoir implémenter un autre algorithme le plus facilement possible si *GREEDY α* devait être remis en cause.

2 Contraintes de fonctionnement et d'exploitation

2.1 Performances

Nous avons des exigences assez hautes sur le temps d'exécution du programme. Effectivement, l'enregistrement des chaînes TV étant quasiment continu, cela demande de pouvoir calculer une solution en très peu de temps (de l'ordre de la seconde) si jamais une quelconque souci venait à invalider la solution trouvée.

Nous cherchons également à maintenir une certaine qualité de solution trouvée. Si l'on compare les solutions trouvées à des solutions optimales, les solutions trouvées ne doivent pas être 8 fois moins pertinentes que les solutions optimales.

2.2 Capacités

Dans notre cas, la capacité de notre algorithme (la taille de l'instance à créer) dépend directement du temps d'exécution. Nous n'avons techniquement pas de limite minimale ou maximale pour la taille de l'instance à créer, il faut juste que l'algorithme s'exécute assez rapidement pour remplir ses fonctions.

Pour notre problème, nous envisageons de créer des instances d'une taille variant de la journée à la semaine s'il n'y a pas décalages entre la diffusion réelle et le programme TV. Dans le cas contraire, il faudra recalculer les quelques prochaines heures pour rectifier la solution.

2.3 Modes de fonctionnement

L'algorithme n'aura qu'un seul mode de fonctionnement, car il n'y aura pas de différence entre la création d'une solution pour une journée ou pour une semaine. La liste de programme TV à résoudre sera tout simplement plus longue et cela ne changera rien au niveau de l'algorithme.

2.4 Contrôlabilité

Il sera important de pouvoir mesurer le temps d'exécution de l'algorithme pour s'assurer de sa performance.

Afin d'assurer la qualité des solutions calculées, l'indice de qualité ρ sera également mesuré avec les poids des intervalles calculés. Cette mesure ne sera effectuée qu'avec des fichiers XMLTV sélectionnés sur lesquels une solution optimale sera pré-calculée afin de pouvoir les comparer.

2.5 Donnés de test

Pour pouvoir effectuer les tests et mesures exprimés précédemment, il nous faut :

- Des fichiers XMLTV de test (Donnés 2020 déjà fournis).
- Des fichiers CSV de parsing python (fichier collection sur les données 2020 déjà fourni).
- Des fichiers CSV de parsing C++ (fichier parsé sur les données 2020 déjà fourni, demande des modifications).
- Des instances déjà résolues de manières optimales (à faire).
- Des instances déjà résolues avec $GREEDY\alpha$ (à faire).
- Des fichiers CSV d'instanciation de la journée (à faire).

2.6 Sécurité

La contrainte de sécurité ne sera pas une forte contrainte pour les raisons suivantes :

Durant la majeure partie du projet, si ce n'est tout le projet, le programme ne tournera que sur ma machine personnelle, ne représentant aucun danger pour la station ou l'école.

L'algorithme ne tournera éventuellement que sur le PC Dell de la station TV s'il y est transféré et n'est donc pas accessible au public. Les utilisations problématiques possibles sont donc également très limitées à ce niveau-là.

Enfin, l'algorithme sera utilisé par du personnel professionnel (comme spécifié plus tôt). Cela limite grandement la possibilité de mauvaise utilisation de l'algorithme. Par mauvaise utilisation, nous entendons par exemple :

— Passage d'une mauvaise liste de programmes ou d'un programme TV erronée.
— Création d'une solution erronée dû à une erreur dans l'algorithme

Dans tous les cas, la seule chose qui est à craindre, c'est de perdre une journée d'enregistrement si une solution d'enregistrement est erronée. Une mauvaise solution ne risque pas d'endommager les cartes d'acquisition ou le PC de la station.

Le programme python va cependant se connecter à internet pour aller chercher les fichiers XMLTV, il faudra s'assurer que l'on utilise toujours de sources fiables. On pourra également filtrer le type de fichiers tolérés.

2.7 Intégrité

Comme le programme doit s'exécuter très vite, le couper pendant l'exécution ne sera pas un problème, car recommencer la procédure ne représentera pas une perte de temps conséquente.

De même, si la station TV est interrompue pendant son enregistrement et qu'elle perd la liste des diffusions à suivre, cela ne créera pas de cas que nous ne gérons pas déjà, car le programme sera déjà fait pour pouvoir se ré-exécuter en cours de journée s'il y a un décalage entre les diffusions du programme TV et la réalité.

3 Hypothèses

Il est possible que malgré les efforts apportés sur l'implémentation, l'instanciation d'une journée se fasse trop lentement. Cela nous forcerait à revoir notre approche.

Il faudra également que les résultats fournis par l'algorithme soient satisfaisants vis-à-vis des poids attitrés. C'est-à-dire que même si l'objectif du ratio de 5.828 est atteint, il faut que la solution fournie soit assez bonne pour pouvoir effectuer des analyses dessus. Dans le cas contraire, l'algorithme utilisé devra être remis en question.

Il se peut que la diffusion des émissions se retrouve être décalée par rapport au programme TV (si une émission prend du retard par exemple). Cela pourrait rendre inexploitable la solution trouvée. Le programme est normalement fait pour pouvoir recalculer la solution si ce problème venait à arriver. En revanche, si cela venait à arriver trop souvent, cela nous forcerait à adopter une approche plus "agile" permettant de prendre des décisions en temps réel.

4 Maintenance et évolution du système

Le calcul de la solution S devra être bien séparé du reste, afin de pouvoir appliquer un autre algorithme que GREEDY si l'une des hypothèses pose souci, ou si une meilleure solution est trouvée. Ainsi, il n'y aura qu'à implémenter le nouvel algorithme et le relier au code de parsing et d'instanciation de la journée.

4.1 Seconde partie de l'algorithme GREEDY

L'article spécifie, dans une seconde partie, comment implémenter une nouvelle solution pour trouver Ci, réduisant la complexité de l'algorithme. Cette solution ne sera explorée qu'une fois que la première version de l'algorithme sera complètement implémentée et les autres fonctionnalités du programme finies.

4.2 Migration PC DELL

Bien que cela ne demandera probablement pas beaucoup de travail, il faudra que le programme soit migré sur la machine pilotant les cartes d'acquisition. Il faudra s'assurer que tout fonctionne bien, mais cela ne constituera pas un projet tout seul, cette tâche pourra être associée avec les autres perspectives d'avenir.

4.3 Boucle automatique

Pour une utilisation pratique de mon algorithme, à l'avenir, il faudra qu'il puisse s'exécuter de manière automatique. La période idéale pour calculer la solution de la journée se trouve entre 4h et 5h du matin, on peut donc souhaiter implémenter une fonctionnalité paramétrable permettant au programme de s'exécuter automatiquement à une heure donnée.

Il faudra cependant toujours garder la main mise sur le calcul des solutions afin de recalculer une nouvelle dans la journée s'il y a un décalage des diffusions. Il est cependant également possible d'imaginer qu'il serait possible d'automatiser le re-calcul de la solution si l'on peut détecter les problèmes de diffusions automatiquement.

4.4 Édition des intervalles

Si l'on continue sur le problème de décalage d'une émission dans la journée, les bases de donnés en ligne ne se mettent pas à jour en temps réel, cela veut dire que même si on re-télécharge le fichier XMLTV en milieu de journée, l'erreur sera toujours présente dans la solution. On peut donc imaginer une fonctionnalité permettant de modifier Ω pour permettre à l'algorithme de re-calculer sa solution.

4.5 Les poids

Pour le moment, les poids associés aux programmes sont attitrés de manière aléatoire dans la partie parsing du programme. Pour la suite, nous souhaiterions pouvoir attitrer ces valeurs manuellement.

4.6 interface

Si le programme est migré sur la machine de la station, il serait envisageable de simplifier son utilisation en créant une interface. Cette idée reste assez secondaire, car elle n'est absolument pas nécessaire pour permettre l'utilisation de l'algorithme implémenté dans la station.

Une interface serait surtout pertinente pour l'éditeur d'intervalles mentionné plus tôt, car son utilisation sera fastidieuse en console de commande.

7 Planification

1 Premier diagramme de Gantt

Voici un premier diagramme de Gantt rapide, créer avant d'avoir eu les réunions de conduite de projet :

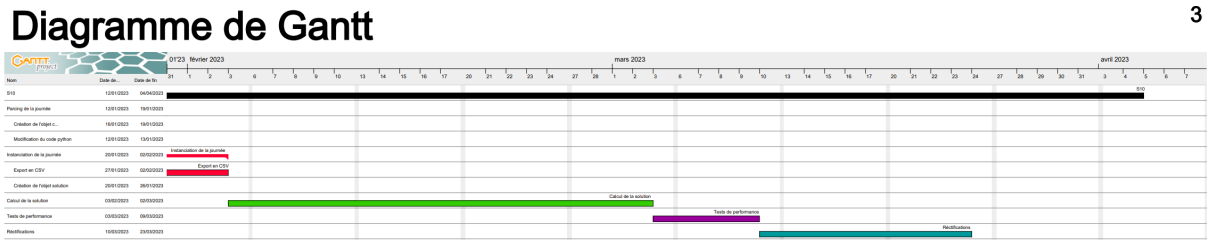


Figure 7.1 – *Premier Gantt*

2 Diagramme de Gantt complet

Une fois l'entretien effectué, j'ai décidé de changer d'outils pour organiser mon travail, j'ai mis en place un Trello avec l'ensemble des tâches effectuées et à venir. Cela m'a permis d'y voir plus clair en le faisait évoluer au cours du semestre 9 et j'ai également créé un nouveau diagramme de Gantt plus complet à partir du Trello :

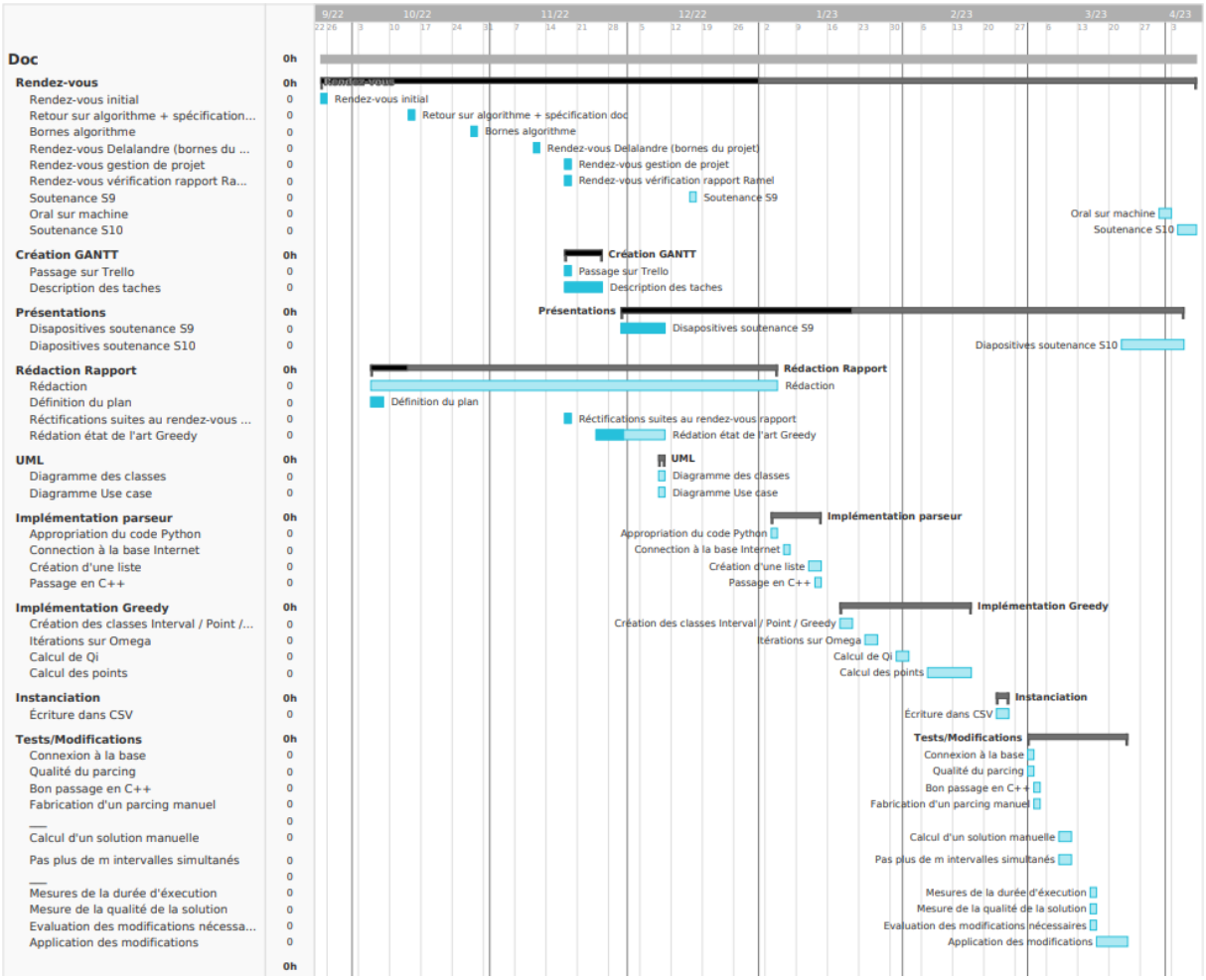


Figure 7.2 – *Gantt Complet*

Ce diagramme sera encor mené à évoluer, mais me donne une bonne liste de tâches à suivre.

3 Déroulement réel

8 Développement

1 Outils utilisés

1.1 Outils de code

J'ai écrit tout le code (C++) du programme en utilisant VS Code avec divers modules. Le code est cependant simplement compilé avec g++ 8.1. Je n'utilise pas de dépendances particulières non plus.

Le code a été versionné régulièrement avec git.

1.2 Documentation

Tout le code a été commenté normalement et de manière assez détaillée. Une partie des commentaires suit la norme Doxygen, la documentation a été générée et est disponible dans le répertoire du projet au format Latex ou HTML.

J'ai également généré les diagrammes UML (UseCase et diagramme de classes) avec Astah.

1.3 Organisation

J'ai répertorié les tâches à effectuer dans un Trello, il s'y trouve :

- Les étapes de rédaction du rapport.
- La création / rectification des diagrammes UML.
- Les rendez-vous.
- Les soutenances.
- Les différentes fonctionnalités à implémenter.
- La période de tests.
- Les mesures à effectuer.
- Les rectifications à apporter.

Je l'ai maintenu à jour tout au long du projet et j'ai essayé de suivre les périodes définies (car ce Trello étain lié à mon diagramme de Gantt).

2 Changements de structure

Des changements ont été effectués entre ce qui était initialement prévu et le rendu final. Nous allons voir ce qui a changé via l'ULM et pourquoi ces changements ont eu lieu

2.1 Diagramme useCase final

J'ai premièrement inclus des manipulations qui n'étaient pas spécifiés dans le diagramme précédent. Il y a surtout la fonctionnalité de récupération de fichier en ligne qui a disparu par manque de temps et, car cette fonctionnalité était très secondaire. Paramétrer l'algorithme est maintenant systématique et linéaire.

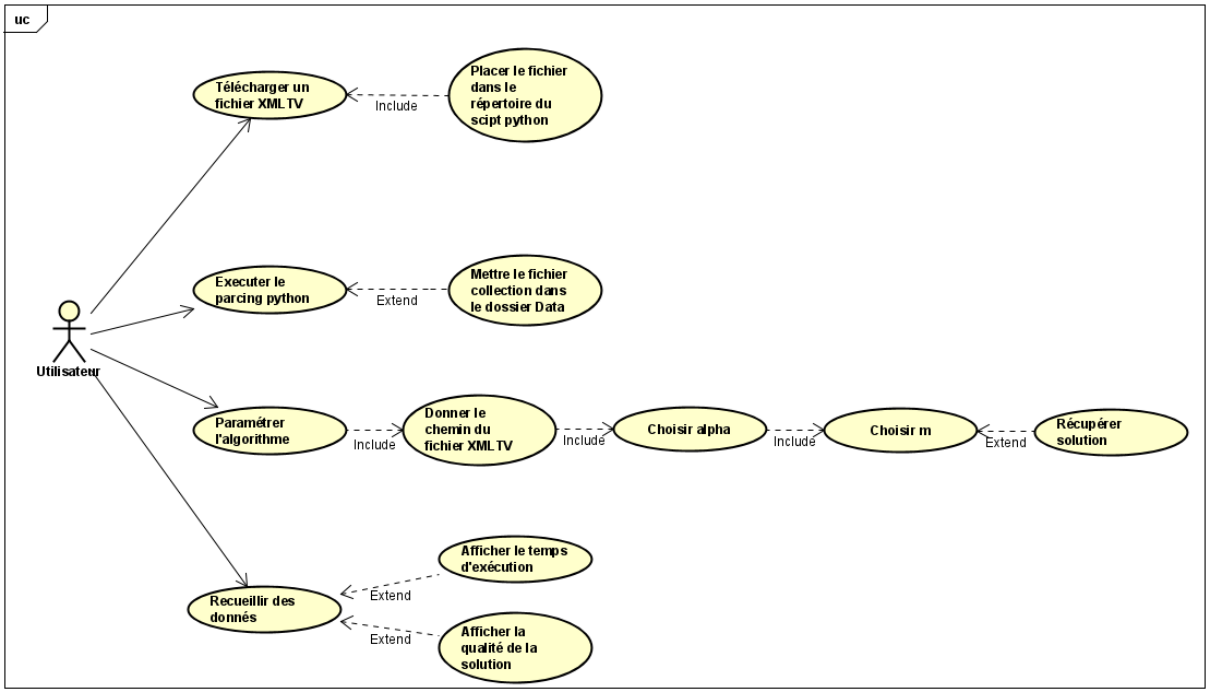


Figure 8.1 – *Diagramme useCase Final*

2.2 Diagramme des classes final

J'ai ajouté les deux parseurs d'entrée et de sortie du programme qui n'étaient pas spécifiés auparavant. On remarque également que beaucoup de types de variables ont été mis à jour. J'utilise beaucoup de vecteurs pour changer leur taille dynamiquement et il n'y a désormais qu'un vecteur d'intervalles, les autres ne sont que des pointeurs pour éviter d'allouer de la mémoire inutilement. La résolution du problème avec Greedy est maintenant découpé en plusieurs fonctions. Enfin, la notion de first et last est passé de la classe interval à la classe Greedy car les first et last ne sont utiles que pendant une boucle et ne doivent pas rester après. Les avoirs dans la classe interval les aurait rendus inutilement persistent.

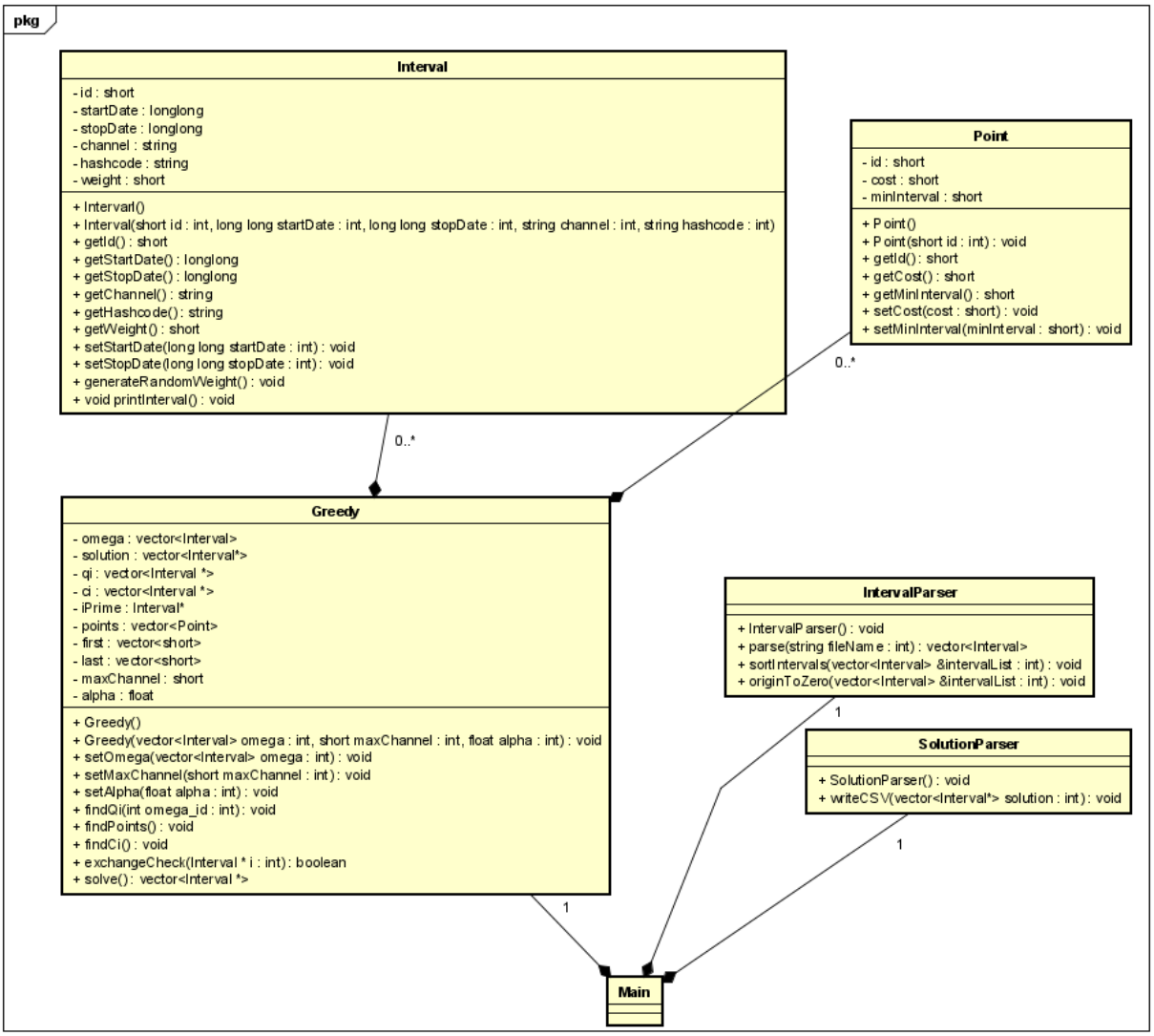


Figure 8.2 – *Diagramme des classes final*

3 Changements d'organisation

Mon diagramme de Gantt a subi quelques changements :

- Modifications du temps prévu pour l'UML : Comme expliqué plus tôt, mon UML initiale n'étais pas du tout assez précis. J'ai donc ajouté du temps.
- Modification du temps de rédaction : je n'avais pas prévu assez de temps pour rédiger la deuxième partie du rapport, j'en ai donc rajouté à la fin
- Modification des priorités : Certaines fonctionnalités n'étaient pas prioritaires, mais je les avais quand même mis au début, car cela suivait un ordre logique de développement. Par manque de temps, je les ai repoussés à la fin.

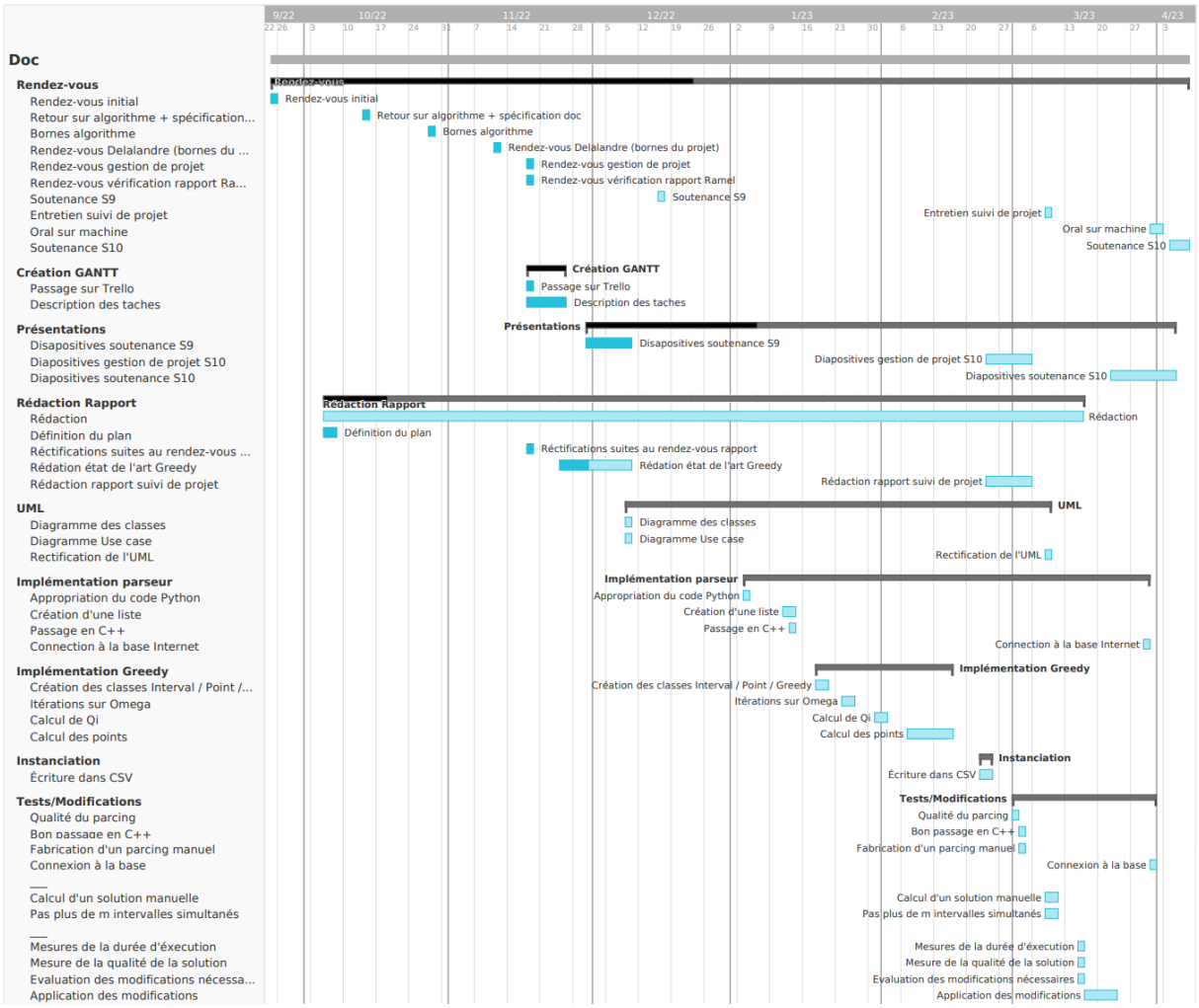


Figure 8.3 – *Diagramme de Gantt mis à jour*

4 Temps d'exécution

Il n'y a jamais eu d'ordre de grandeur précis à respecter en ce qui concerne le temps d'exécution, il faut juste éviter de perdre trop de temps d'enregistrement si l'algorithme doit être appelé. J'ai fait des teste sur un parsing et calcul sur une période d'un mois et l'exécution du programme reste dans l'ordre de la seconde sur ma machine personnel. La machine de la station TV étant plus puissante, il semblerait que cette contrainte soit respectée.

9 Bilan et conclusion

1 Bilan du semestre 9

La liste des tâches à effectuer au S9 est accomplie, c'est-à-dire que tout ce qui concerne la documentation est effectué :

- Rédaction du rapport.
- Compréhension et explication des articles fournis.
- Application de l'algorithme pour explication.
- Création des graphes UML.
- Création des diagrammes de Gantt

Cela dit, j'ai mis beaucoup de temps à bien comprendre l'article sur *Greedy*α et à correctement l'expliquer. Finalement, je reste dans les temps et j'ai même eu le temps de commencer un peu le développement avant le S10, mais j'aurais aimé plus avancer.

2 Bilan du semestre 10

Pour la partie développement, l'algorithme Greedy a été implémenté dans sa quasi-totalité. La possibilité de connecter le programme à une base de fichiers XMLTV n'a pas été implémenté par manque de temps.

Pour la partie tests, des tests ont été effectués, mais méritent d'être approfondies. Je vais notamment créer un fichier collection à la main après les soutenances et séparer les tests en cas de tests plus fins.

Pour la mesure des performances, la mesure du temps d'exécution est faite, mais il manque la mesure de la qualité de la solution. La résolution de ce problème sera liée à la création d'un fichier collection type.

3 Bilan sur la qualité

L'une des contraintes à respecter était la complexité de l'algorithme que je pense avoir bien réussi à respecter. J'ai également plutôt évité les allocations de mémoire inutiles. Les manquements sur la qualité se trouvent plutôt sur les parties finales, c'est-à-dire les tests et les mesure.

4 Bilan auto-critique

Le négatif : Il y a eu une certaine quantité de changements au niveau des UML, il faut que j'arrive à mieux me projeter sur le code lors de la création de l'UML. Il va de même pour le diagramme de Gantt, j'ai pour le moment beaucoup de mal à évaluer la durée des taches.

Je m'y suis pris trop tard pour les tests. J'ai encore du mal à appliquer les bonnes pratiques de tests et j'ai peu de maitrises des outils nécessaires.

Le positif : J'ai réussi à commenter mon code tout le long, aussi bien en commentaires normaux qu'en commentaires Doxygen.

Je pense également avoir fait des réunions assez régulières avec mon encadrante et elles ont toujours été pertinentes. J'ai également toujours gardé une trace écrite des réunions.

Annexes

A Cahier de test

Les tests visent à garantir l'exactitude, l'intégrité, la sécurité et les performances du logiciel.

1 Tests simples

Comme je l'ai expliqué plus tôt, je me suis pour le moment limité pour le moment à un simple tableau de tests

Cahier de tests

Fonctionalité	Entrée	Attendu	Priorité	Status Implémentation	Status test
Parseur d'intervalles					
Connexion à la base en ligne	Entrée d'une URL	Téléchargement d'un fichier XMLTV	Basse	Non	
Parsing des intervalles (Python)	fichier XMLTV	Un fichier collection		déjà implémenté	Validé
Parsing des intervalles (C++)	fichier collection	Un vecteur d'intervalles parsés	Haut	Oui	validé
Mise de l'origine à 0	Un vecteur d'intervalles parsés	La date de début du premier intervalle à été soustraite à tous les intervalles	Moyenne	Oui	validé
Intervalle					
Génération des poids	Un intervalle initialisé	Un poids aléatoire généré à partir du hashcode	Haute	Oui	Validé
Affichage de l'intervalle	Un intervalle initialisé	Ue ligne affichant toutes les informations	Basse	Oui	Validé
GREEDYα					
Calculer Qi	Un vecteur d'untervalles avec un intercvalle "I" à insérer	Un sous ensemble du vecteur d'intervalle contenant tous les intervalles en conflit avec "I"	Haute	Oui	Validé
Trouver I'	Un vecteur d'untervalles avec un intercvalle "I" à insérer	Un doublon I' de I déjà présent dans la solution	Haute	Oui	Validé
Calculer les points	Un sous-ensemble Qi	Un vecteur de points sans couts / Un vecteur first et last	Haute	Oui	Validé
Calculer Ci	Qi + vecteurs de points + vecteur first + vecteur last	Les couts mis à jours pour les points, ainsi qu'une trace de l'intervalle à cout minimal par points	Haute	Oui	Validé
Mettre à jour la solution	Ci + vecteur de points	remplace Ci (avec I') par I si l'échage est bon en fonction des poids	Haute	Oui	Validé
Fonction solve	Le vecteur de tous les intervalles "Omega"	Appel toutes les fonctions en boucle pour générer une solution	Moyenne	Oui	Validé
Parseur de solution					
Ecrire dans CSV	Un vecteur d'intervales	Un fichier CSV contenant des intervalles de temps à enregistrer sur tel chaine	Haute	Oui	Validé
Main					
Entrée des donnés	Un utilisateur utilisant le programme	récupération du fichier solution / valeurs "m" et "alpha"	Moyenne	Oui	Validé
Execution auto du script	Un utilisateur utilisant le programme	L'appel au script python est fait automatiquement	Basse	Non	

Figure A.1 – *Cahier de tests*

B Documentation

1 Documents disponibles

Sera disponible dans l'archive de rendu de projet :

- Un README expliquant comment se servir du programme et comment le recompiler si besoin.
- La documentation Doxygen disponible en HTML ou Latex.
- Le cahier de tests.
- Les diagrammes UML en fichier astah.

2 README

Voici le contenu du README qui fait office de manuel d'utilisation :

Utilisation :

- Télécharger un fichier XMLTV.
- Exécuter le scripte python : /Python/1_generate_latency.py pour générer le fichier collection. (Le script python lit tous les fichiers XML dans data/01, c'est ici qu'il faut placer les fichiers XMLTV)
- Lancer l'exécutable c++.
- Donner le chemin d'accès au fichier collection (fichiers exemples présents dans data).
- Donner le nombre d'intervalles simultanés souhaités (nombre de cartes d'acquisition pour la station TV).
- Donner la valeur d'alpha (ratio appliqué aux poids pour l'échange d'intervalles).
- Le fichier solution se trouvera dans le dossier Solution

Compilations : Pour re-générer l'exécutable, simplement utiliser g++ sur main.cpp en indiquant les classes au linker : g++ main.cpp Greedy.cpp Point.cpp Interval.cpp IntervalParser.cpp SolutionParser.cpp

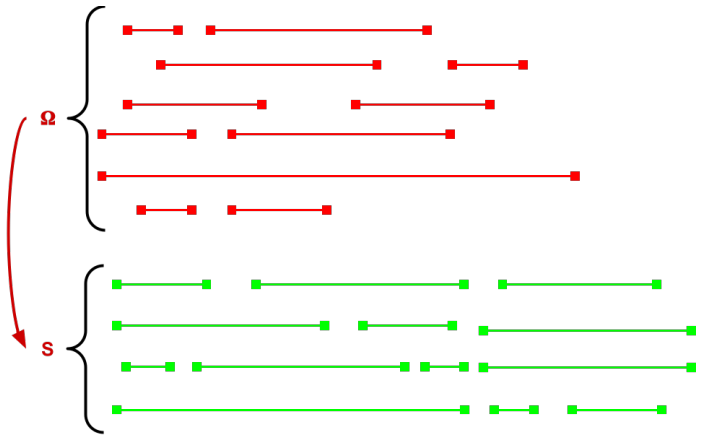
Des lignes d'affichages sont commentées avec "// [PRINT] //", vous pouvez les utiliser.

Projet Recherche & Développement

Optimisation pour la sélection de programmes TV :
Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer

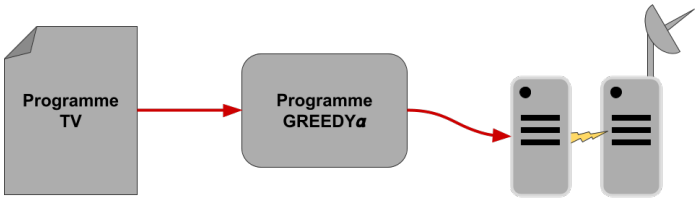
Corentin Lezé–Robert
Encadrement : Tifenn Rault

En collaboration avec Polytech



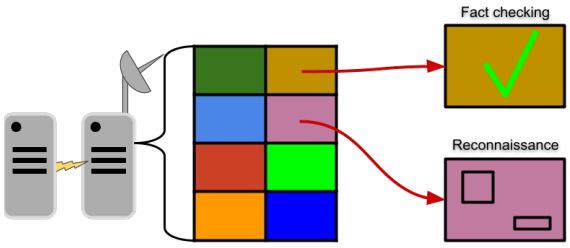
Objectifs

- Calculer une liste de programmes à enregistrer
- Implémanter un algorithme Greedy
- Effectuer le calcul rapidement



Mise en œuvre

1. Parser les donnés des fichiers XMLTV
2. Implémanter Greedy en C++
3. Parser la solution pour être exploitable par la station



Résultats attendus

Un programme permettant de calculer rapidement une solution à partir d'un fichier XMLTV permettant de fournir une solution à une station TV ne pouvant enregister que 8 programmes simultanéments.

Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Projet Recherche & Développement

Optimisation pour la sélection de programmes TV : Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer

Corentin Lezé–Robert

Encadrement : Tifenn Rault

En collaboration avec Polytech

Objectifs

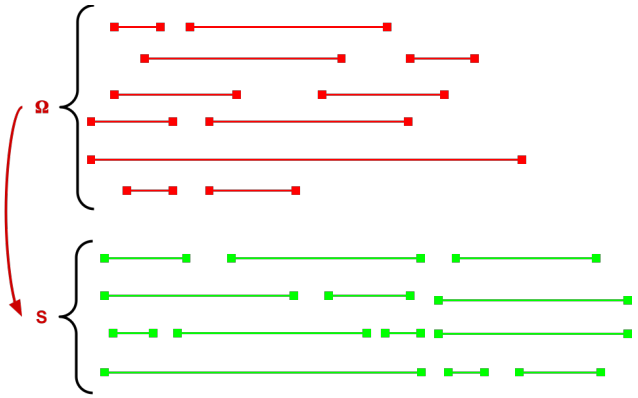
- Calculer une liste de programmes à enregistrer
- Implémanter un algorithme Greedy
- Effectuer le calcul rapidement

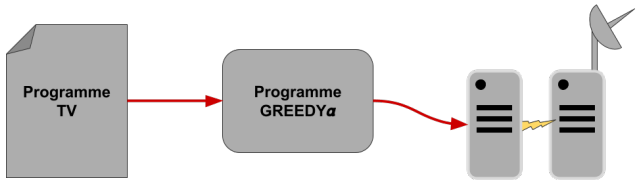
Mise en œuvre

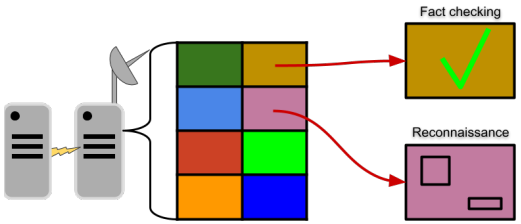
1. Parser les donnés des fichiers XMLTV
2. Implémanter Greedy en C++
3. Parser la solution pour être exploitable par la station

Résultats attendus

Un programme permettant de calculer rapidement une solution à partir d'un fichier XMLTV permettant de fournir une solution à une station TV ne pouvant enregister que 8 programmes simultanéments.







Ecole Polytechnique de l'Université de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Optimisation pour la sélection de programmes TV

Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer

Résumé

Ce rapport traite du projet R&D Root 5 "Algorithmes d'optimisation pour la sélection de programmes TV à enregistrer". Dans ce rapport est expliqué dans un premier temps le contexte et les enjeux liés à la station TV, l'algorithme choisi pour répondre à la solution ainsi que son état de l'art afin de donner un aperçu global du projet. Dans un second temps, les détails tel que les interfaces, les fonctionnalités et les spécifications non fonctionnelles seront abordées. Enfin, les détails liés aux développement seront expliqués.

Mots-clés

C++, Python, sélection, Station TV, Complexité

Abstract

This report is an analysis of the Root 5 project "Optimisation algorithm to select TV programs to record". In this report, will be first explained the context and stakes due to the TV Station, the chosen algorithme used as an awnser to our problem and the state of the art linked to it in order to give a global overview of the project. Then, details such as interfaces, fonctionalities and non fonctional specifications will be described. Lastly, we will go over developpement details.

Keywords

C++, Python, Selection, TV Station, Complexity

Entreprise
Polytech

Tuteur entreprise
Mathieu Delalandre

Étudiant
Corentin Lezé–Robert (DI5)

Tuteur académique
Tifenn Rault