



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](http://polytech.univ-tours.fr)

## Projet Recherche & Développement 2021-2022

# Développement d'une heuristique dédiée pour la planification de la maintenance corrective des trains au sein de la SNCF



**POLYTECH<sup>®</sup>**  
**TOURS**

### Entreprise

SNCF - Pôle ingénierie du matériel de Saint Pierre des Corps



### Tuteur entreprise

Romuald DETRUE

### Étudiant

Vincent LE GARJAN (DI5)

### Tuteurs académiques

Ronan BOCQUILLON

Vincent T'KINDT

# Liste des intervenants

## Entreprise

SNCF - Pôle ingénierie du matériel de Saint Pierre des Corps  
56 Avenue Pompidou  
37700 Saint Pierre des Corps, France  
[www.sncf.com/fr/reseau-expertises/direction-du-materiel](http://www.sncf.com/fr/reseau-expertises/direction-du-materiel)



Nom	Email	Qualité
Vincent LE GARJAN	<a href="mailto:vincent.legarjan@etu.univ-tours.fr">vincent.legarjan@etu.univ-tours.fr</a>	Étudiant DI5
Ronan BOCQUILLON	<a href="mailto:ronan.bocquillon@univ-tours.fr">ronan.bocquillon@univ-tours.fr</a>	Tuteur académique, Département Informatique
Vincent T'KINDT	<a href="mailto:vincent.tkindt@univ-tours.fr">vincent.tkindt@univ-tours.fr</a>	Tuteur académique, Département Informatique
Romuald DETRUE	<a href="mailto:romuald.detrue@sncf.fr">romuald.detrue@sncf.fr</a>	Tuteur entreprise



# Avertissement

Ce document a été rédigé par Vincent Le Garjan susnommé l'auteur.

L'entreprise SNCF - Pôle ingénierie du matériel de Saint Pierre des Corps est représentée par Romuald Detruie susnommé le tuteur entreprise.

L'Ecole Polytechnique de l'Université de Tours est représentée par Ronan Bocquillon et Vincent T'kindt susnommés les tuteurs académiques.

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

L'auteur reconnaît 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 reconnaît qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable des tuteurs académiques 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

Vincent Le Garjan, *Développement d'une heuristique dédiée pour la planification de la maintenance corrective des trains au sein de la SNCF*, Projet Recherche & Développement, Ecole Polytechnique de l'Université de Tours, Tours, France, 2021-2022.

```
@mastersthesis{
  author={Le Garjan, Vincent},
  title={Développement d'une heuristique dédiée pour la planification de la maintenance
    corrective des trains au sein de la SNCF},
  type={Projet Recherche \& Développement},
  school={Ecole Polytechnique de l'Université de Tours},
  address={Tours, France},
  year={2021-2022}
}
```



# Table des matières

Liste des intervenants	<b>a</b>
Avertissement	<b>b</b>
Pour citer ce document	<b>c</b>
Table des matières	<b>i</b>
Table des figures	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1 Acteurs, enjeux et contexte .....	1
Présentation du projet.....	1
Acteurs principaux .....	2
2 Objectifs.....	2
3 Hypothèses .....	2
4 Bases méthodologiques.....	3
<b>2 Description générale</b>	<b>4</b>
1 Environnement du projet .....	4
2 Caractéristiques des utilisateurs.....	4
3 Fonctionnalités du système .....	5
4 Structure générale du système.....	5
<b>3 État de l'art</b>	<b>7</b>
1 Heuristique d'ordonnancement et ordonnancement ferroviaire .....	7
2 Précédents travaux et projets en cours .....	7
Projet Recherche & Développement de B. Pouvreau .....	7

	Stage de fin de Master de V. Jourdan .....	8
	Projet Recherche & Développement de M. Bervet .....	8
	Stage 4A de L. Perdereau .....	8
	Projet Recherche & Développement de T. Ray .....	9
	Projet Recherche & Développement de V. Le Garjan .....	9
3	Heuristique dédiée .....	9
	Construction du graphe de flot maximum à coût minimum .....	9
	Arbre de recherche .....	10
	Un petit exemple de fonctionnement .....	10
<b>4</b>	<b>Analyse et conception</b> .....	<b>12</b>
1	Analyse .....	12
1.1	Besoin dans le détail .....	12
1.2	Spécifications .....	12
2	Modélisation proposée .....	12
<b>5</b>	<b>Mise en oeuvre</b> .....	<b>15</b>
1	Outils et librairie utilisés .....	15
2	Éléments d'implémentation .....	15
2.1	La classe Instances .....	16
2.2	InstanceGenerator.cpp .....	16
3	Interface d'utilisation .....	16
<b>6</b>	<b>Bilan et conclusion</b> .....	<b>18</b>
1	Bilan du semestre 9 .....	18
2	Bilan du semestre 10 .....	18
3	Bilan sur la qualité .....	18
4	Bilan auto-critique .....	19
	<b>Annexes</b> .....	<b>20</b>
<b>A</b>	<b>Planification, gestion de projet</b> .....	<b>21</b>
1	Livrables et deadlines .....	21
2	Diagramme de Gantt .....	22
3	Suivi des tâches .....	24
<b>B</b>	<b>Description des interfaces</b> .....	<b>26</b>
1	Interfaces matérielles/logicielles .....	26
2	Interfaces homme/machine .....	26
3	Interfaces logicielles/logicielles .....	26

<b>C</b>	<b>Cahier de Spécifications</b>	<b>27</b>
1	Spécifications fonctionnelles .....	27
1.1	Générateur d'instance.....	27
	Générer une instance.....	27
	Afficher les instances .....	27
	Générer un fichier JSON .....	27
2	Spécifications non fonctionnelles .....	28
2.1	Contraintes de développement et conception .....	28
2.2	Contraintes de fonctionnement et d'exploitation.....	28
2.2.1	Performances.....	28
2.2.2	Capacités .....	28
2.2.3	Contrôlabilité .....	28
2.2.4	Sécurité .....	28
<b>D</b>	<b>Cahier du développeur</b>	<b>29</b>
1	Introduction .....	29
2	Architecture du projet .....	29
2.1	Les librairies .....	29
2.2	La classe Instances.....	30
2.3	InstanceGenerator.cpp .....	30
3	Descriptions détaillées de données exploitées .....	30
3.1	Les instances .....	30
3.2	Les JSON .....	31
4	Descriptions détaillées des classes, modules, réalisations .....	32
4.1	La classe Instances.....	32
4.2	Le fichier main : InstanceGenerator.cpp .....	32
<b>E</b>	<b>Document d'installation</b>	<b>35</b>
<b>F</b>	<b>Document d'utilisation</b>	<b>36</b>
<b>G</b>	<b>Cahier de test</b>	<b>37</b>
<b>H</b>	<b>Explication détaillée de l'heuristique dédiée</b>	<b>39</b>
<b>I</b>	<b>Exemple de fonctionnement de l'heuristique dédiée</b>	<b>45</b>
<b>J</b>	<b>Bibliographie</b>	<b>51</b>
<b>K</b>	<b>Glossaire</b>	<b>52</b>
<b>L</b>	<b>Acronymes</b>	<b>53</b>



# Table des figures

<b>1</b>	<b>Introduction</b>	
1.1	Photo d'une rame Z50000 de la SNCF .....	1
<b>2</b>	<b>Description générale</b>	
2.1	Schéma explicatif de l'architecture du système .....	4
2.2	Diagramme des cas d'utilisation du système .....	5
2.3	Schéma explicatif du fonctionnement du système .....	6
<b>3</b>	<b>État de l'art</b>	
3.1	Exemple explicatif d'une planification : attribution d'une rame à une voie.....	8
3.2	Graphe pour l'explication du fonctionnement général de l'heuristique dédiée .....	10
3.3	Arbre pour l'explication du fonctionnement général de l'heuristique dédiée.....	10
3.4	Exemple de graphe de l'heuristique dédiée .....	11
3.5	Exemple d'arbre de l'heuristique dédiée .....	11
<b>4</b>	<b>Analyse et conception</b>	
4.1	Diagramme de classes réalisé par M. Bervet .....	14
<b>5</b>	<b>Mise en oeuvre</b>	
5.1	Diagramme de la classe Instances.....	16
<b>A</b>	<b>Planification, gestion de projet</b>	
A.1	Livrables et deadlines .....	21

A.2	Diagramme de Gantt final - partie 1 .....	22
A.3	Diagramme de Gantt final - partie 2 .....	23
A.4	Suivi de tâches - partie 1 .....	24
A.5	Suivi de tâches - partie 2 .....	25
 <b>B Description des interfaces</b>		
B.1	Schéma présentant l'organisation du logiciel d'ordonnancement .....	26
 <b>D Cahier du développeur</b>		
D.1	Diagramme de la classe InstanceDataFormatter .....	30
D.2	Diagramme de la classe InstanceData.....	31
D.3	Diagramme de la classe Instances.....	32
D.4	Diagramme du format JSON d'instance - partie 1 .....	33
D.5	Diagramme du format JSON d'instance - partie 2 .....	34

# 1

## Introduction

### 1 Acteurs, enjeux et contexte

#### Présentation du projet

La **Société Nationale des Chemin de fer Français (SNCF)** fait rouler 11 000 trains transportant 5 millions de voyageurs par jour. 70% de ces voyageurs se déplacent en Île de France dont une partie sur les lignes H et K du Transilien. Sur ces lignes roulent des **rames** Z50000 qui ont la particularité de pouvoir remonter en temps réel les anomalies repérées sous forme de signalement à un centre de contrôle. Celles-ci peuvent être de nature diverses impliquant un criticité différente : une porte qui ferme mal, un élément cassé, un problème moteur ...



**Figure 1.1** – Photo d'une rame Z50000 de la SNCF

Jusqu'à maintenant, ces trains "malades" (qui ont émis un signalement), étaient traités à la main par un agent qui s'employait donc à trouver un créneau libre pour la rame et l'orientait sur une voie de maintenance comportant l'infrastructure nécessaire à la réparation. Mais, il y a quelques années, la SNCF décide de remplacer cette tâche fastidieuse par un logiciel d'aide à la décision qui s'occuperait d'**ordonnancer** les opérations de maintenance signalées.

C'est pourquoi elle a fait appel en 2018 à l'université de Tours et, plus particulièrement, au **Laboratoire d'Informatique Fondamentale et Appliquée de Tours (LIFAT)** afin de concevoir et développer un outil de recherche opérationnelle permettant de résoudre son problème. Cet outil sera ensuite intégré dans le logiciel d'aide à la décision développé par l'entreprise LGM.

## Acteurs principaux

Ce projet a vu passer plusieurs étudiants que sont Benjamin Pouvreau, Valentine Jourdan, Maël Bervet, Loris Perdereau et Tom Ray et dont le travail est détaillé [un peu plus loin](#). Il est encadré par :

- Romuald DETRUE (Surveillant Etat Technique de niveau 2 et chef du projet pour la direction d'ingénierie du matériel à la SNCF) - **Client**
- Ronan BOCQUILLON & Vincent T'KINDT (Encadrants de ce "Projet Recherche & Développement" à Polytech' Tours) - **Maitrise d'ouvrage (MOA)**

La partie détaillée dans ce rapport de **Projet Recherche & Développement (PRD)** est réalisée par :

- Vincent LE GARJAN (Étudiant en 5ème année au département informatique de Polytech' Tours) - **Maitrise d'œuvre (MOE)**

## 2 Objectifs

Toutes les  $x$  unités de temps (tous les jours, tous les 3 jours, toutes les semaines, ... par exemple), un agent de la SNCF récupère les différents signalements émis par les rames. Ces signalements préviennent une panne ou une casse quelconque et possède une criticité pouvant aller de 1 pour un petit désagrément à 6 pour des pannes très graves. En fonction de cette criticité, les rames doivent être réparées dans un délai plus ou moins long, avant donc une date dite "due", et cela sur une voie possédant les infrastructures permettant la dite réparation.

Le problème est donc d'ordonnancer la maintenance de ces trains sur les différentes voies afin de minimiser les retards des réparations par rapport aux dates dues. Cependant les rames en question possèdent un **planning de roulement** qu'elles doivent respecter. Les voies peuvent aussi déjà être en partie occupées par des maintenances correctives ou préventives. L'ordonnancement d'une rame nécessite donc un creux de roulement et un créneau de voie libre au même moment assez long pour effectuer la maintenance demandée.

À cela, d'autres contraintes s'ajoutent, notamment le fait que la quantité de rames pouvant entrer sur un site de maintenance est limitée par période. Aussi, si une opération ne peut pas être ordonnancée, on essaye tout de même de faire au moins la partie "diagnostique" de cette maintenance et de repousser (on dit "rejeter") cette opération à plus tard.

L'objectif de ce Projet Recherche & Développement est donc de terminer de développer et d'implémenter une **heuristique** afin de résoudre le problème posé par la SNCF. Il s'agit de produire un algorithme permettant de donner une planification de la maintenance corrective des trains satisfaisant les contraintes énoncées.

## 3 Hypothèses

Une contrainte supplémentaire a été énoncée par la SNCF. Celle-ci décrit le fait que des rames peuvent effectuer un **croisement** entre une rame saine allant faire une maintenance préventive et une rame "malade". Cette contrainte est apparue car les croisements simplifient l'organisation des maintenances. Cependant, dans un premier temps, cette contrainte ne sera pas prise en compte.

On fait aussi l'hypothèse, pour l'instant, que les cas où il y a de nombreuses rames devant effectuer une maintenance au même moment sur la même voie est assez rare pour l'ignorer.

## 4 Bases méthodologiques

Pour ce projet, différents outils de gestion de projet ont été mis en place. Un gestionnaire de versioning (Git / Gitlab) est utilisé et un diagramme de Gantt a été créé.

Pour la rédaction des documents, nous nous sommes servi d'outils comme la suite Microsoft Office et notamment de OneNote ou encore Overleaf pour l'utilisation de LaTeX. Les diagrammes ont été réalisés sur Draw.io.

L'implémentation de l'heuristique se fera en C++ grâce à l'**Integrated Development Environment (IDE)** Microsoft Visual Studio.

Des réunions fréquentes sont organisées avec Ronan Bocquillon et Vincent T'Kindt, les encadrants du projet (MOA). Ces temps servent à échanger autour de l'avancement du projet, à répondre aux questionnements et à réorienter si besoin vers les points importants. Plus fréquemment encore, des échanges de mail ont lieu et des rapports sont livrés, dans le même but. Ainsi notre méthodologie peut s'apparenter à une méthodologie de type "agile".

# 2

## Description générale

### 1 Environnement du projet

Comme nous avons déjà pu le décrire, ce projet s'inscrit dans une suite de projets menés par différents étudiants de Polytech' Tours. Ces différents projets seront détaillés dans la partie "**État de l'art**". Tous se sont attelés à créer un outil de résolution, exact ou approché, pour la planification de la maintenance des trains. Cet outil est intégré dans un logiciel déjà existant, développé par LGM et connecté à l'infrastructure de la SNCF, utilisé par des agents au travers d'un outil WEB interne.

L'objectif est de développer et d'implémenter une heuristique permettant de résoudre ce problème. Elle remplacera l'outil de résolution exact pour l'instant en place dans le logiciel d'aide à la décision afin de se séparer des contraintes liées à l'utilisation d'un outil propriétaire.

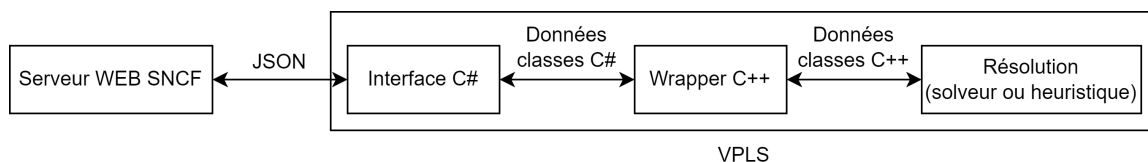


Figure 2.1 – Schéma explicatif de l'architecture du système

### 2 Caractéristiques des utilisateurs

Les utilisateurs du système sont des agents de la SNCF. Ceux-ci sont actuellement chargés, entre autres tâches, de planifier la maintenance des trains des lignes H et K à la main. Ce logiciel les aidera donc dans ce travail.

Aucune connaissance particulière ni aucune formation n'est nécessaire pour utiliser le logiciel d'aide à la décision en dehors de la manipulation classique d'un ordinateur. Un simple bouton permettra de récupérer la planification à partir de l'instance alors automatiquement envoyée.

Quant au sujet précis de se Projet Recherche & Développement, il s'agit du noyau de résolution du logiciel. Les utilisateurs du système n'y auront donc pas directement accès.

### 3 Fonctionnalités du système

Lorsqu'un utilisateur le demande, le système logiciel récupère une instance JSON générée par le serveur WEB de la SNCF et l'envoie au noyau de résolution : un solveur CPLEX renvoyant une résolution exacte comme actuellement ou l'implémentation de l'heuristique que nous allons terminer dans ce PRD. Enfin, le système retourne le(s) résultat(s) de planification.

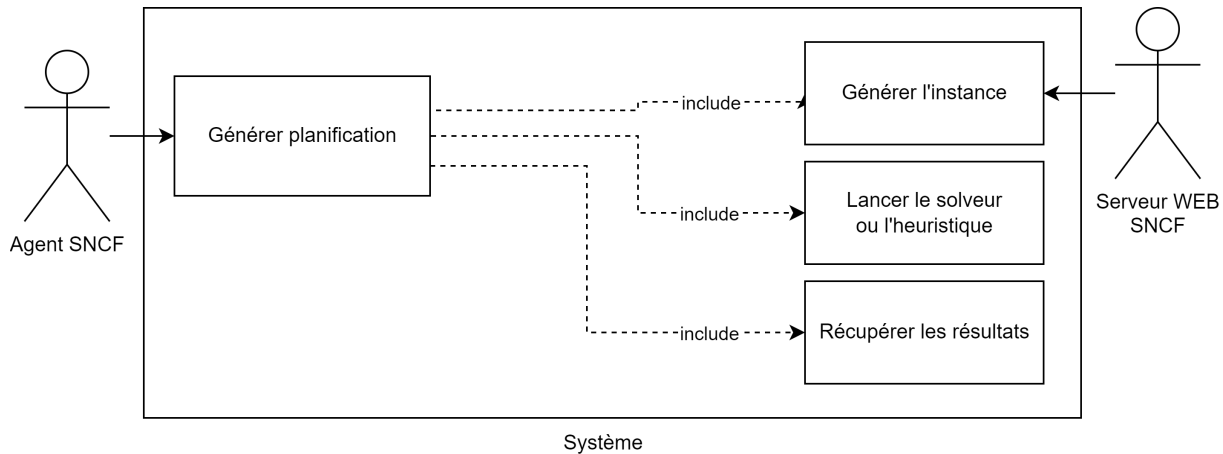


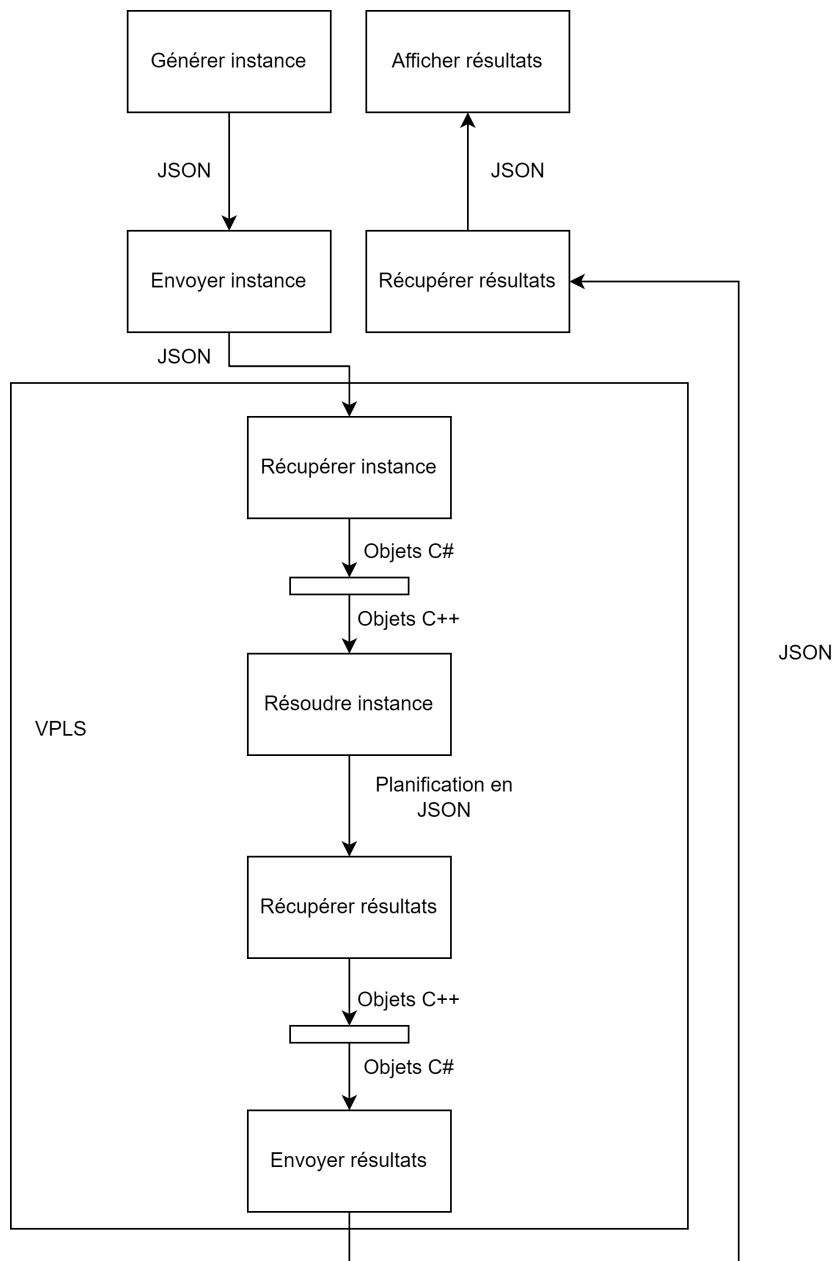
Figure 2.2 – Diagramme des cas d'utilisation du système

### 4 Structure générale du système

Le système se décompose en 2 grandes parties.

D'une part, il y a le serveur WEB géré par la SNCF. Cette partie du système permet de générer, à partir des données de la SNCF, des instances, au format JSON, comportant toutes les informations nécessaires à la planification de la maintenance des rames : les creux de roulement des rames, les positions des sites de maintenances, les disponibilités des voies, les signalements, ... Ce fichier JSON est ensuite envoyé à la seconde partie du système.

De l'autre côté, il y a le logiciel "VPLS", créé par LGM et l'université de Tours. Celui-ci récupère les données JSON avant de les transformer en objets C# puis ces objets C# en objets C++ par l'intermédiaire d'un "wrapper". Ces objets sont ensuite, pour l'instant, transférés à un solveur CPLEX donnant une résolution exacte. L'objectif de ce PRD, nous l'aurons compris, sera de remplacer ce noyau par une heuristique. La planification résultante sera renvoyée au "wrapper" puis retransmise au serveur WEB sous format JSON.



**Figure 2.3** – Schéma explicatif du fonctionnement du système

# 3

## État de l'art

### 1 Heuristique d'ordonnancement et ordonnancement ferroviaire

Il existe de nombreux papiers présentant des solutions heuristiques à des problèmes d'ordonnancement. Il existe aussi quelques travaux proposant des solutions à des problèmes d'ordonnancement ferroviaires, notamment pour la maintenance des voies. Mais il n'existe pas, à ma connaissance après avoir effectué quelques recherches, de papiers présentant une résolution heuristique pour un problème d'ordonnancement de maintenance corrective de rames en dehors des travaux précédemment effectués sur ce même projet. Nous avons donc décidé, en accord avec la maîtrise d'ouvrage, de ne pas nous intéresser d'avantages à la littérature de ce domaine mais plutôt aux précédents travaux qui ont été réalisés dans le cadre de ce projet de logiciel d'aide à la décision pour la SNCF.

### 2 Précédents travaux et projets en cours

La SNCF souhaite donc créer un logiciel permettant la planification de la maintenance des trains. Elle a fait appel à l'entreprise LGM qui s'est occupé d'implémenter une application permettant de lier les outils de la SNCF à l'outil de recherche opérationnelle dont le développement a été confié à Polytech' Tours depuis 2018. Concernant ce second point, différents projets se sont ainsi attelés à résoudre le problème.

#### Projet Recherche & Développement de B. Pouvreau

Durant son Projet Recherche & Développement [4], Benjamin Pouvreau a commencé par analyser et modéliser le problème mathématique et les contraintes associées en un problème de **Programmation Linéaire Standard (PLS)** afin de développer le premier algorithme de résolution. Le principe de ce modèle est d'utiliser des variables binaires afin de définir si telle rame est affectée à telle voie tout en minimisant le retard des opérations en fonction de leur gravité. Cet algorithme de résolution exacte prenait appui sur le solveur CPLEX.

Ce projet a aussi permis de modéliser l'architecture de l'application d'aide à la décision dans sa globalité.

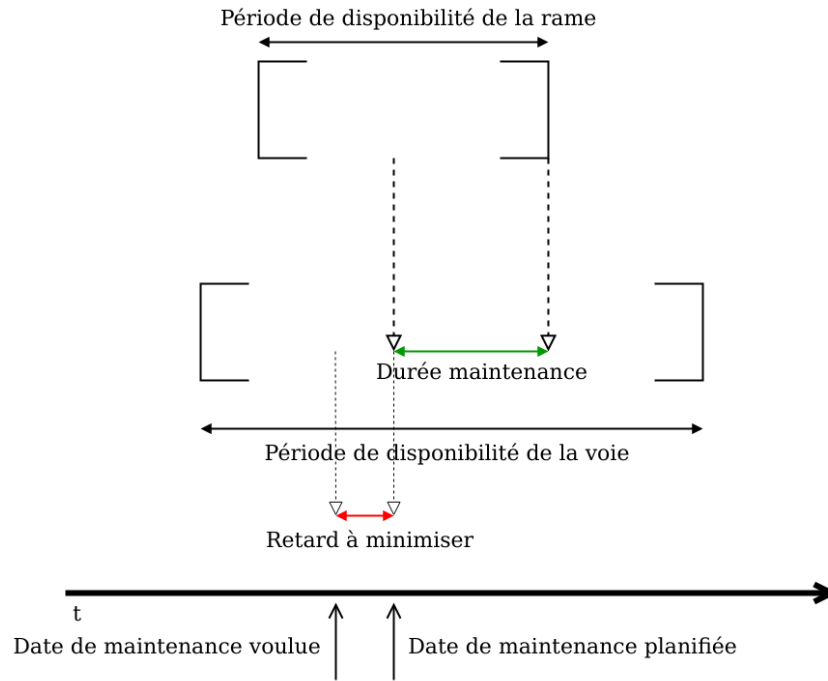


Figure 3.1 – Exemple explicatif d'une planification : attribution d'une rame à une voie

#### Stage de fin de Master de V. Jourdan

Ce projet [2], réalisé par Valentine Jourdan dans le cadre de son stage de fin de Master, étend les travaux de Benjamin Pouvreau en modélisant le même problème mais en utilisant la **Programmation Par Contraintes (PPC)**. Une solution permettant de générer des instances réalistes a aussi été créée dans le même temps afin de pouvoir tester le(s) modèle(s).

#### Projet Recherche & Développement de M. Bervet

Les méthodes de résolution exactes choisies jusqu'à maintenant utilisent un solveur propriétaire. Afin de s'émanciper des contraintes légales qui en incombent à la SNCF, il a été décidé de concevoir une heuristique [6] permettant de résoudre le problème de façon approchée. Mael Bervet s'est donc occupé de spécifier l'algorithme et de commencer à l'implémenter [1] lors de son Projet Recherche & Développement.

Cette heuristique étant le cœur du projet actuel, nous prendrons le temps d'en reparler de façon plus approfondie, un peu plus tard, dans la section "**Heuristique dédiée**".

#### Stage 4A de L. Perdereau

Suite à l'apparition d'une nouvelle contrainte spécifiée par la SNCF, Loris Perdereau a été chargé, dans le cadre de son stage de 4<sup>ème</sup> année, d'adapter le modèle mathématique afin de prendre en compte la contrainte des croisements [3]. Celle-ci décrit la possibilité de croiser des rames, c'est à dire d'échanger leur trajet, plutôt que de créer un nouveau trajet.

Il a été émis l'hypothèse que les croisements sont généralement peu nombreux. Cette contrainte impliquant trop de modifications sur le modèle de base, il a donc été décidé de plutôt modifier les instances (les données d'entrée) en calculant les croisements à l'avance.

### Projet Recherche & Développement de T. Ray

Dans le cadre de son Projet Recherche & Développement [5], réalisé en parallèle de celui-ci, Tom Ray cherche à terminer une première version du logiciel d'aide à la décision utilisant la résolution mathématique exacte afin de faire tester une version fonctionnel de ce logiciel aux agents de la SNCF. Il pourra ensuite développer la brique permettant de calculer les croisements et de les ajouter au données d'entrée.

### Projet Recherche & Développement de V. Le Garjan

Ce projet, dont ce document est le rapport, est la continuation du PRD de Maël Bervet. L'objectif est de prendre en main et de compléter l'heuristique dédiée puis de récupérer la première implémentation de cette heuristique afin de la corriger et de l'achever. Enfin, il s'agira d'intégrer cette implémentation dans le logiciel d'aide à la décision de la SNCF, à la place du noyau de résolution exacte.

## 3 Heuristique dédiée

Le cœur de ce Projet Recherche & Développement est construit autour de l' "heuristique dédiée". Cet algorithme a été créé par Maël Bervet avec l'aide de Vincent T'kindt, Ronan Bocquillon et Valentine Jourdan dans le cadre de son **Projet Recherche & Développement**. Nous allons ici résumer son fonctionnement. Pour plus de détails, une version descriptive de l'heuristique est disponible en annexe.

Cette heuristique est séparée en deux grandes parties. Une première partie a pour objectif de construire un graphe de flot maximum à coût minimum permettant de calculer le coût d'une affectation d'un créneau "rame" à un créneau "voie". La deuxième partie de cette heuristique cherche à créer un ordonnancement, grâce à un arbre de recherche, en définissant quelles solutions d'affectation sont les meilleures pour chaque rame, en commençant par celles qui ont les signalements les plus critiques.

On peut noter que cette heuristique ne prend pas en compte les croisements. Elle comporte aussi quelques problèmes dans le traitement de certains cas. Ces deux points seront traités dans une seconde partie, lorsque l'heuristique actuel aura une implémentation fonctionnelle.

### Construction du graphe de flot maximum à coût minimum

Pour construire ce graphe de flot maximum à coût minimum, nous plaçons deux sommets "source" et "puits" aux extrémités du graphe. C'est le flot traversant le graphe entre ces sommets qui représentera la quantité de temps passé sur les différentes opérations ainsi que leur rejet ou non. Nous pourrions alors en calculer le coût.

On relie ensuite à la source  $S$  les différents sommets  $Op_i$  représentant les différents signalements donc les différentes opérations à réaliser. Ces opérations peuvent soit être ordonnancées et donc être effectuées sur un créneau de la rame (symbolisé par le sommet  $I_k^{(r,i)}$ ), soit être rejetées à plus tard (à un temps hors de la période d'ordonnancement, symbolisé par le sommet  $R_i$ ).

Les sommets de rejet sont reliés au puits  $P$  (car non ordonnancés). Les sommets "intervalle voie", quant à eux, sont reliés aux sommets "intervalle rame"  $I_k^{(v,j)}$  dont la durée de l'intersection entre ces deux intervalles est supérieur à la durée de diagnostic. C'est à dire qu'on relie un sommet "intervalle rame" et un sommet "intervalle voie" si et seulement si l'intersection entre ces deux intervalle laisse le temps à la rame de se faire diagnostiquer.

Enfin, les sommets "intervalle voie"  $I_k^{(v,j)}$  sont reliés aux sommets "intervalle site"  $I_k^{(s,l)}$  correspondants et les sommets "intervalle site" au puits  $P$ .

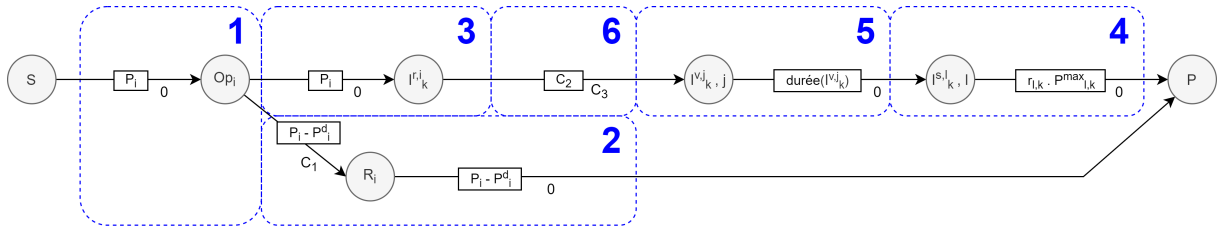


Figure 3.2 – Graphe pour l'explication du fonctionnement général de l'heuristique dédiée

### Arbre de recherche

Une fois le graphe de la partie précédente construit, celui-ci va être utilisé afin d'affecter des rames à des voies et d'en calculer le coût. La décision d'affectation sera faite grâce à un arbre de recherche.

Pour chaque opération, en commençant par les plus longues parmi les plus critiques, nous allons tester l'affectation, complète ou en diagnostique, à chaque intervalle voie relié. Nous allons donc faire passer le flot correspondant à la durée totale et le flot correspondant à la durée en diagnostique dans les différents arcs "intervalle voie" - "intervalle rame" correspondant au sommet opération en question. Cette opération permet de tester la faisabilité et, si faisable, le coût de l'affectation d'un rame à un intervalle "voie", représentée par un nœud dans l'arbre.

Nous pouvons alors évaluer ces nœuds de l'arbre. Parmi les nœuds faisables, nous prenons les  $n$  nœuds les plus prometteurs, c'est à dire avec le coût le plus faible. Nous passons ensuite à l'opération suivante et nous effectuons ainsi de suite jusqu'à avoir ordonné toutes les opérations.

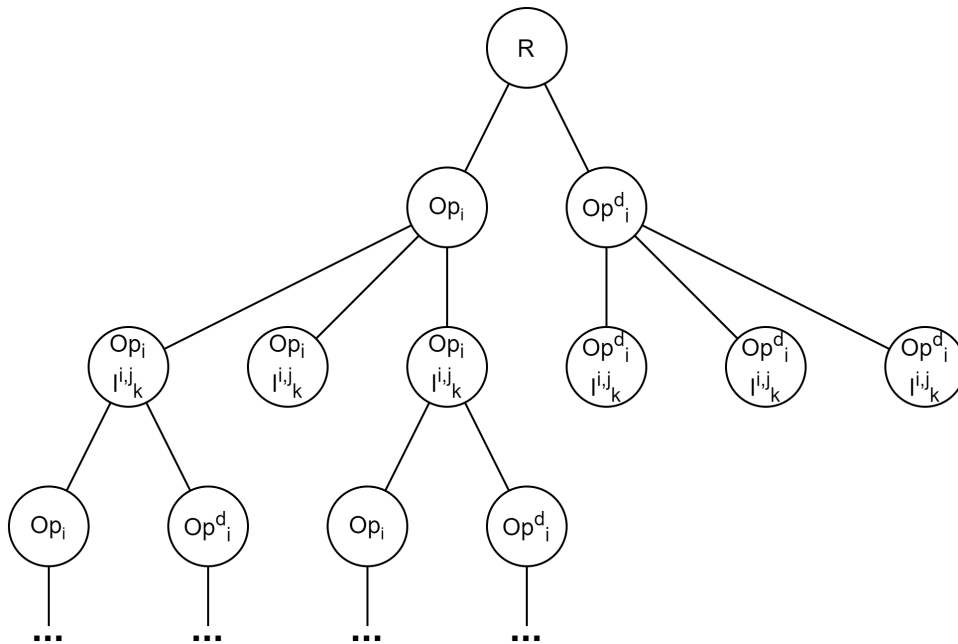
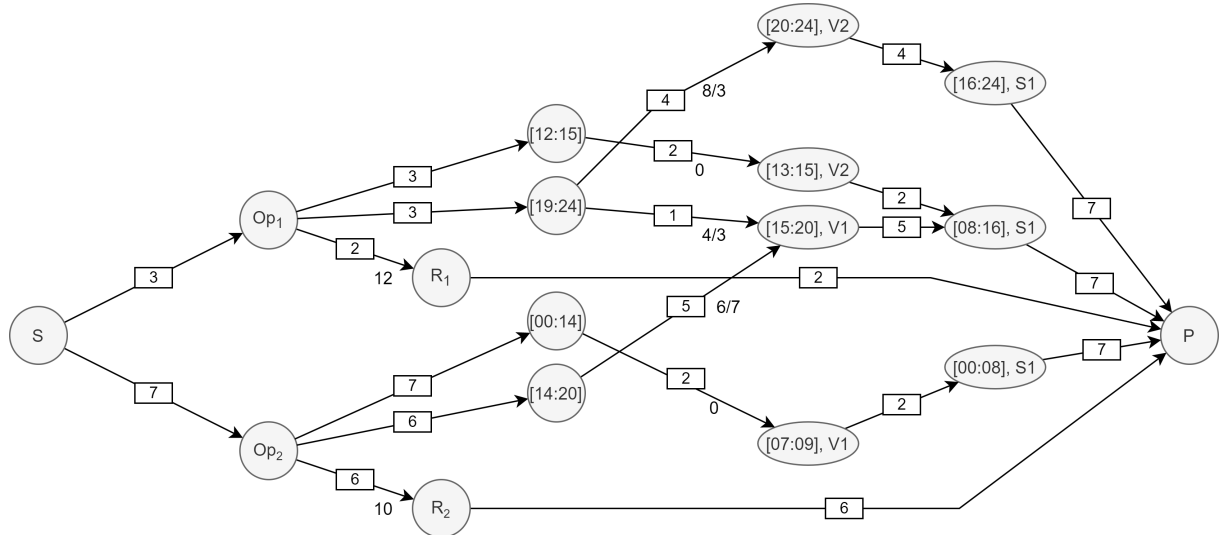


Figure 3.3 – Arbre pour l'explication du fonctionnement général de l'heuristique dédiée

À la fin de la construction de cet arbre, nous connaissons l'affectation de chaque rame à une voie, que ce soit pour la réparation complète, ou juste pour un diagnostique.

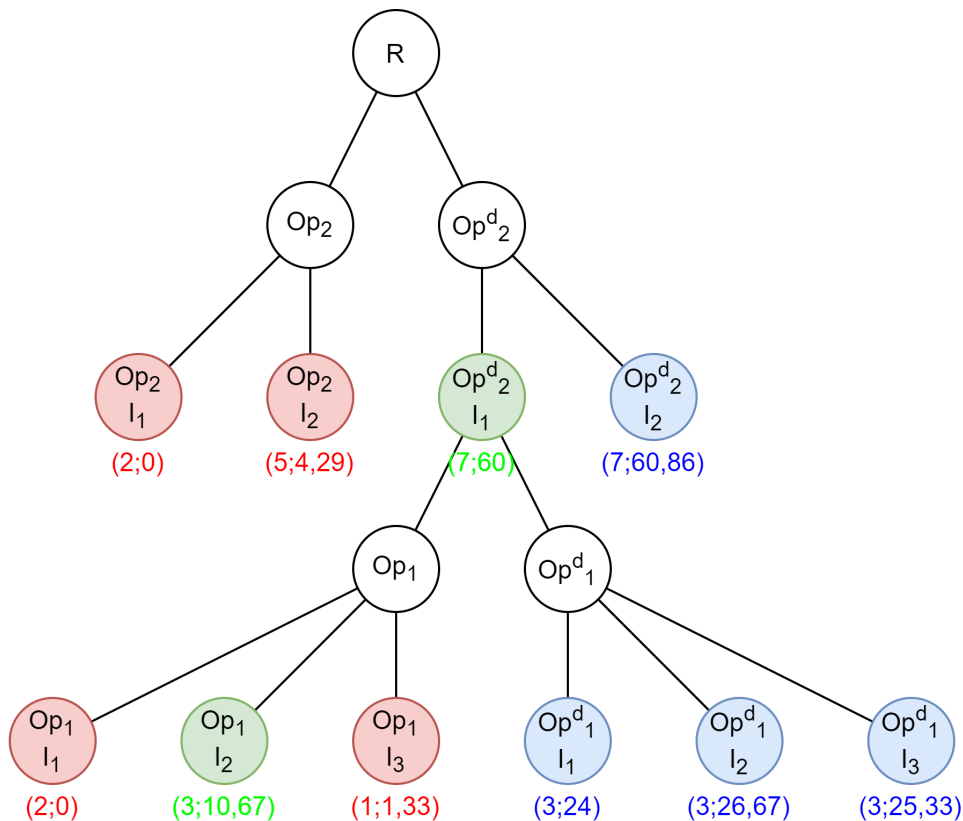
### Un petit exemple de fonctionnement

Pour comprendre le fonctionnement de l'heuristique dédiée, rien de mieux qu'un petit exemple d'exécution. Voici, ci-dessous, le graphe et l'arbre obtenus avec une instance comportant 2 rames, 3 voies et 1 site de maintenance.



**Figure 3.4** – *Exemple de graphe de l’heuristique dédiée*

Après avoir suivi l'algorithme de construction du graphe de flot maximum à coût minimum, nous obtenons le resultat ci-dessus et pouvons donc lancer la deuxième partie : la construction de l'arbre de recherche. Sur le schéma ci-dessous, les nœuds rouges sont infaisables, les nœuds verts sont les nœuds choisis. La première valeur correspond au flot en sortie (doit être égale au flot d'entrée pour une solution faisable) ; la seconde valeur donne le coût du nœud.



**Figure 3.5** – *Exemple d’arbre de l’heuristique dédiée*

La meilleure affectation ici est donc de faire passer l'opération 2 en diagnostique sur l'intervalle "voie-rame" 1 et l'opération 1 complète sur l'intervalle "voie-rame" 2.

# 4

## Analyse et conception

### 1 Analyse

#### 1.1 Besoin dans le détail ...

Faisons quelques rappels. La SNCF souhaite avoir un logiciel permettant d'ordonnancer la maintenance corrective de ces rames roulant sur les lignes H et K du Transilien, et ce afin d'orienter les agents vers d'autres tâches. Car, en effet, cette planification est pour l'instant réalisée par des agents de la SNCF. Ceux-ci récupèrent une liste des rames ayant besoin d'une maintenance et les répartissent sur les voies permettant cette maintenance à un horaire convenable (pour la voie et la rame). L'objectif est donc avec ces mêmes données, de proposer une répartition qui soit la plus optimale possible.

Plus précisément, l'objectif de ce projet sera de terminer d'implémenter l'heuristique dédiée permettant l'ordonnancement, revue dans un premier temps lors du semestre 9, dans le logiciel de la SNCF.

#### 1.2 Spécifications

L'objectif de ce projet est donc de reprendre une implémentation, assez aboutie, et de la compléter, de la déboguer afin d'obtenir une version fonctionnelle. Cette implémentation a été réalisée en C++ sous Visual Studio et utilise des bibliothèques particulières telles que Boost. Il faudra aussi garder en tête un aspect performance et capacité, bien que ceux-ci ne soient pas très contraignants, puisque le programme doit retourner une solution dans un temps raisonnable.

### 2 Modélisation proposée

L'essentiel du travail consistera à reprendre une implémentation déjà bien entamée. Cependant, ayant reçu le début d'implémentation de l'heuristique par M. Bervet que peu de temps avant le rendu de ce rapport, je n'ai pas encore eu le temps de l'étudier. Cela se fera donc dans un second temps, lors du semestre 10.

Nous pourrions donc vraiment modéliser l'implémentation de l'heuristique qu'après avoir pris en main la partie réalisée par M. Bervet. Voici tout de même un diagramme de classes réalisé qu'il a créé présentant l'état supposé de l'application à l'heure actuel.

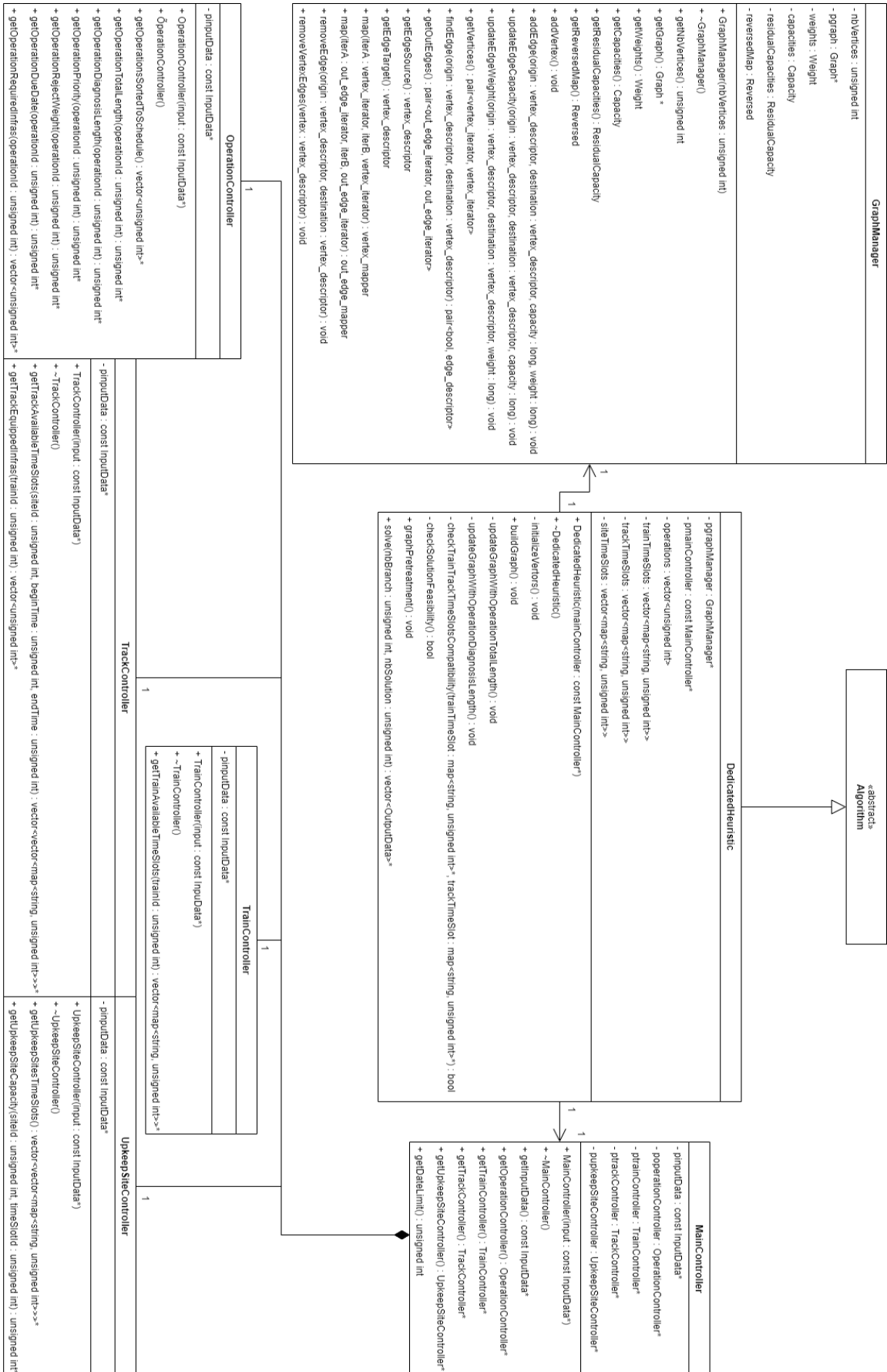


Figure 4.1 – Diagramme de classes réalisé par M. Bervet

# 5

## Mise en oeuvre

L'objectif de ce projet était de reprendre et terminer l'implémentation d'une heuristique dédiée pour la maintenance corrective des trains. Malheureusement, durant ce PRD, j'ai subi une période de trou qui ne m'a pas permis d'emmener ce projet à son terme. La partie mise en oeuvre relatera donc la partie que j'ai eu le temps de réaliser, à savoir : l'extraction et l'implémentation d'un générateur d'instances.

### 1 Outils et librairie utilisés

L'implémentation de ce générateur d'instances a été réalisée en C++ sous Visual Studio puisque la partie "génération aléatoire" a été extraite d'un projet déjà existant ("VPLS Univ"), implémenté en C++ par Valentine Jourdan. D'autres outils tels que Git et GitLab ont pu être utiles notamment pour faire du versionning.

Du côté des librairies, deux externes ont été utilisées :

- **effolkronium/random** en version 1.4.0. Elle permet de générer des nombres aléatoires ou de gérer l'aléatoire en général de façon simple. Cette librairie était déjà utilisée dans le code des classes extraites du projet VPLS, c'est pourquoi nous nous sommes permis de l'utiliser ici aussi.
- **nlohmann/json** en version 3.10.5. Elle permet de créer des objets JSON manipulables très simplement.

Ont aussi été importées les classes développées par Valentine Jourdan et permettant la génération d'instances, notamment les classes "InstanceData" représentant une instance et ses données ainsi que "InstanceDataFormatter" permettant de générer aléatoirement l'instance. Elles sont intégrées dans le projet du générateur d'instance comme une sorte de librairie en "boîte noire" puisqu'il n'y avait aucun intérêt à y toucher.

### 2 Éléments d'implémentation

Le logiciel se divise en trois grandes parties :

- Les **librairies** (voir la section précédente et dans le **cahier du développeur**)
- La classe **Instances**
- Le fichier **InstanceGenerator.cpp** (fichier main)

## 2.1 La classe Instances

La classe **Instances** a été développée pour permettre de gérer les instances plus facilement et notamment les objets `InstanceData` : les objets générés par le générateur de VPLS-Univ. Cette classe comprend 4 attributs :

- **instancesDatas** : un vecteur d'instances (ou plutôt d'objets "InstanceData")
- **criticitiesDatas**, **infrastructuresDatas** et **infrastructuresFrequencies** : des données fixes quel que soit l'initialisation de l'objet permettant de générer des instances cohérentes

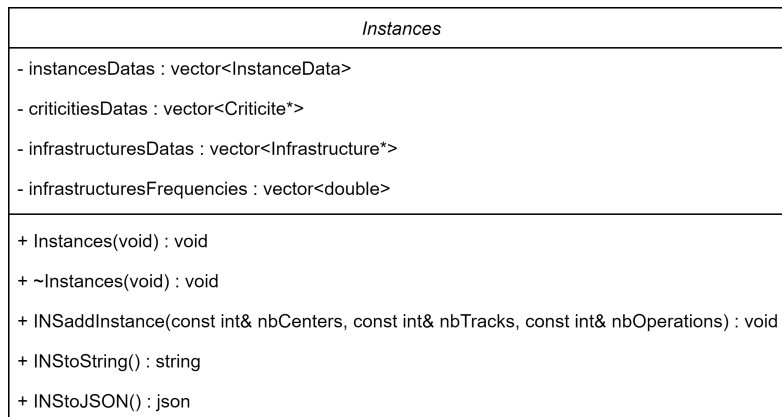


Figure 5.1 – Diagramme de la classe Instances

Cette classe comporte aussi plusieurs méthodes. Parmi celles-ci on retrouve évidemment un constructeur par défaut et un destructeur permettant notamment d'initialiser et de détruire les attributs paramétrant la génération des instances.

Les trois dernières méthodes représentent enfin la partie la plus importante de la classe Instances. La méthode **INSaddInstance** permet de générer aléatoirement une instance (un objet `InstanceData`) avec  $x$  sites de maintenance,  $y$  voies par site et  $z$  opération à réaliser ;  $x$ ,  $y$  et  $z$  étant entrés en paramètres. Pour cela, la méthode utilise la méthode `createAndFormat()` de la classe `InstanceDataFormatter`. Cette instance est alors ajoutée au vecteur des instances de la classe Instances, l'attribut `instancesDatas`.

La méthode **INStoString** permet de retourner une chaîne de caractères comportant toutes les données de toutes les instances présentes dans le vecteur d'instances.

La méthode **INStoJSON** permet de retourner un objet JSON (de la librairie `nlohmann/json`) au bon format lisible et interprétable par le projet qui résout le problème d'affectation et contenant toutes les instances.

## 2.2 InstanceGenerator.cpp

Le fichier **InstanceGenerator.cpp** fait office de fichier main. Il récupère les paramètres entrés par l'utilisateur et crée les instances en fonction avant de les enregistrer dans le (ou les) fichier(s) JSON souhaités. Pour cela, il crée une instantiation de la classe Instances et, pour les différents paramètres entrés, itère la méthode `INSaddInstance`. A la fin, il ne lui reste plus qu'à utiliser la méthode `INStoJSON` et enregistrer l'objet JSON obtenu dans le (ou les) fichier(s) qui convient.

## 3 Interface d'utilisation

Le générateur d'instance s'utilise simplement en ligne de commande à condition d'y accoler les bons paramètres :

**InstanceGenerator.exe** <fichier\_entree> <fichier\_sortie> <multi-JSON>

Les paramètres à indiquer :

- <fichier\_entree> est à remplacer par le nom du fichier de paramètres. Si le fichier n'existe pas, un fichier portant ce nom sera créé avec des paramètres par défaut. Il est aussi possible d'indiquer "" pour utiliser les paramètres par défaut sans créer de fichier de paramètres.
- <fichier\_sortie> doit être substitué par le nom du fichier de sortie sous le format <nom>.json
- <multi-JSON> indique l'emplacement ou doit être indiqué si l'ont souhaite séparer les différentes instances générées en plusieurs fichiers JSON (1) ou enregistrer toutes les instances dans un même fichier JSON (0).

# 6

## Bilan et conclusion

### 1 Bilan du semestre 9

Lors de ce semestre, la tâche principale a été de comprendre le sujet et le travail existant concernant les outils de planification de la maintenance corrective. J'ai pu prendre en main l'heuristique dédiée, en corriger quelques éléments et noter ses limites actuelles afin de les effacer plus tard si le temps le permet. Des documents permettant de redécrire précisément le fonctionnement de cette heuristique ont été rédigés.

Dans la fin de ce semestre 9 et pendant le semestre 10, l'objectif sera de reprendre l'implémentation en C++ de l'heuristique réalisée par M. Bervet et de la continuer afin de pouvoir ensuite l'intégrer dans le logiciel d'aide à la décision. Il faudra compléter de façon plus approfondie les spécifications. Des tests seront aussi à prévoir.

### 2 Bilan du semestre 10

Pendant ce second semestre de PRD, et du fait de la période de trou, l'objectif initial de terminer l'implémentation de l'heuristique dédiée n'a pas été mené à terme. Les attendus ont donc alors été reciblés et les tâches ont été redéfinies vers l'extraction et le développement dans un projet séparé du générateur d'instances. Cette mission a été menée à bien puisqu'un projet a été créé à partir du générateur d'instance auparavant présent dans le projet "VPLS Univ" développé par Valentine Jourdan, et le nouveau générateur d'instances extrait est fonctionnel.

Cependant, il reste encore plusieurs travaux avant d'avoir une heuristique développée et utilisable. Il faut encore modifier l'interface pour que le projet récupère en entrée un JSON et plusieurs tests et debugages sont nécessaires pour terminer le projet "heuristique dédiée".

### 3 Bilan sur la qualité

À la fin de ce projet, le générateur d'instances a bien été extrait du projet "VPLS Univ" dans un nouveau projet. Il permet de générer, selon des paramètres entrés par l'utilisateur, des instances correctement lisibles et interprétables par la partie résolution (voir le [cahier de tests](#)). L'archive livrée comprend toutes les informations, les fichiers de code, les exécutables nécessaires à la maintenabilité, à la portabilité ainsi qu'à la bonne continuité et reprise du projet.

## 4 Bilan auto-critique

Lors de la première partie de ce projet, je pense avoir réussi à mettre en pratiques mes connaissances en recherche opérationnelle et mes capacités d'analyse pour comprendre et synthétiser l'heuristique dédiée dans un document explicatif.

La seconde partie de ce projet s'est un peu moins bien déroulé que la première puisque une période de trou au milieu a empêcher la réalisation de tous les objectifs initiaux. Même si j'ai eu l'occasion de développer un projet en C++, cette issue est assez décevante pour moi puisque ce projet dans sa globalité est très intéressant et j'aurais aimé aller plus loin.

Cependant, participer à ce projet m'a permis de développer et mettre en pratique de nombreuses compétences apprises tout au long de mon parcours post-bac et notamment pendant mes 3 années de parcours "ingénieur" au département informatique. Des spécifications aux tests, en passant par l'état de l'art, la gestion de projet, le développement, la documentation, j'ai pu expérimenter et me confronter à toutes les étapes d'un projet. Pour chacune d'entre elles, il m'a fallu mettre en oeuvre ce que j'avais appris mais aussi apprendre de l'expérience en cours. Enfin, j'ai aussi et surtout pu, pour la première fois, participer à un projet assez grand (puisque sur plusieurs années et avec plusieurs acteurs) et comprendre son fonctionnement.

# Annexes

# A

## Planification, gestion de projet

### 1 Livrables et deadlines

Voici les dates des livrables et deadlines.

LIVRABLES & DEADLINES	Dates
Document explicatif sur l'heuristique dédiée v2 (partie "graphe")	07/10/2021
Document explicatif sur l'heuristique dédiée v3 (partie "graphe")	14/10/2021
Exemple de déroulé de l'heuristique dédiée v1 (partie "graphe")	
Résumé hebdomadaire	
Document explicatif sur l'heuristique dédiée v4	20/10/2021
Exemple de déroulé de l'heuristique dédiée v3	
Compte-rendu réunion 2021-11-18	25/11/2021
Résumé hebdomadaire v2	
Rapport S9 version présentation	10/12/2021
Rapport S9 version finale	03/01/2021
Rapport "Suivi de projet"	14/03/2021
Rapport Final	03/04/2021

Figure A.1 – Livrables et deadlines

2 Diagramme de Gantt

Voici le diagramme de Gantt final.

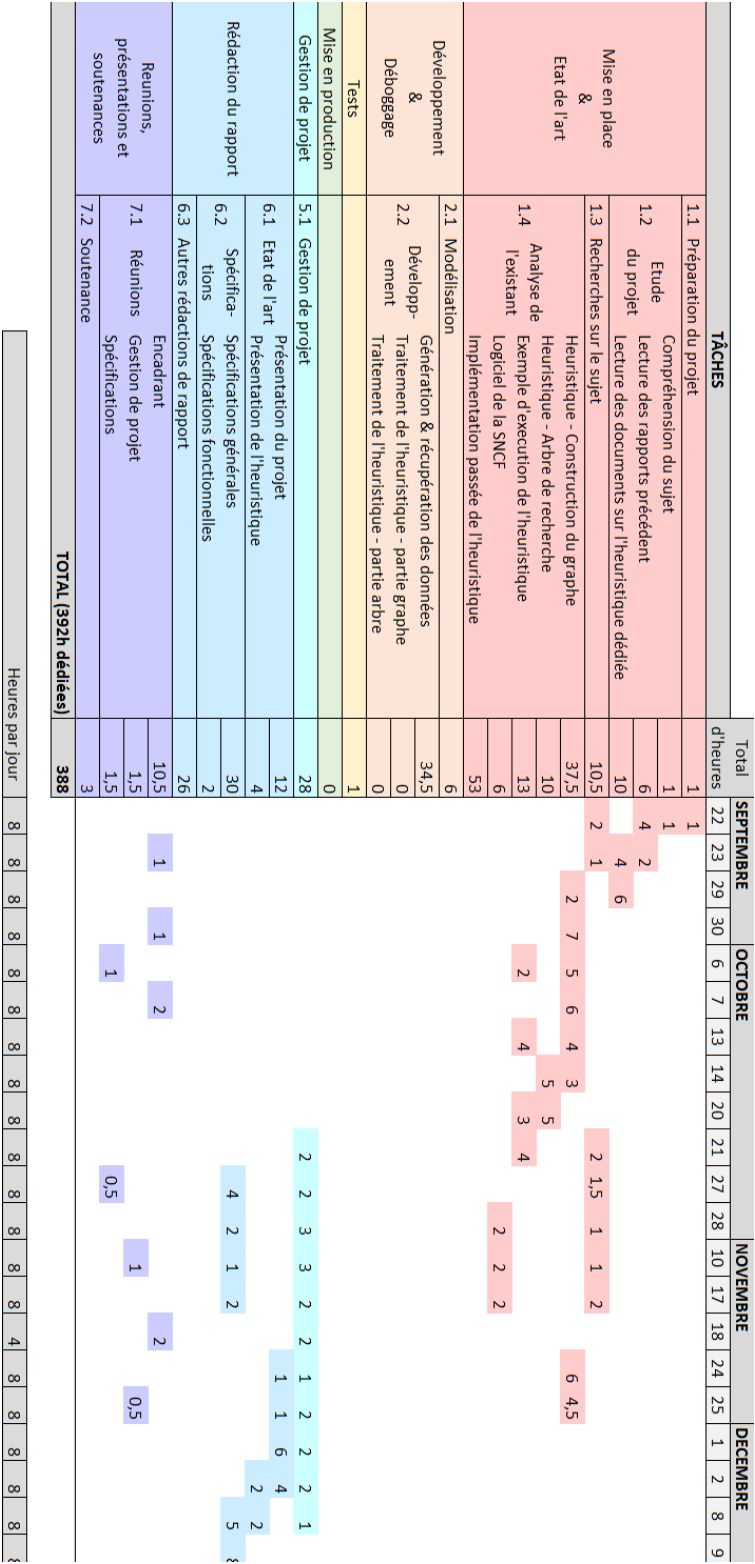


Figure A.2 – Diagramme de Gantt final - partie 1

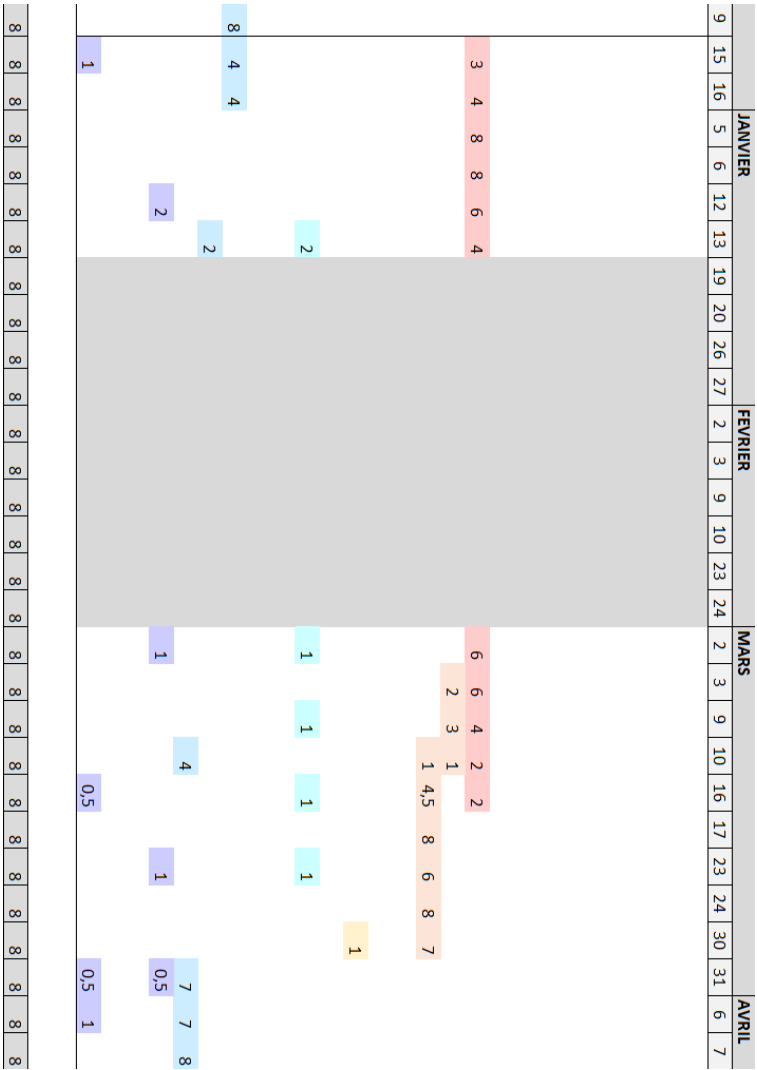


Figure A.3 – Diagramme de Gantt final - partie 2

### 3 Suivi des tâches

Voici le tableau de suivi des tâches.

Séance	Notes
22/09/2021	Lecture des rapports de PRD et de Master de POUVREAU Benjamin, JOURDAN Valentine et BERVET Maël. Recherches rapides pour un début d'état de l'art.
23/09/2021	Réunion de présentation avec R. Bocquillon : rappel et réexplications du sujet, premières orientations de lecture. Suite lecture du rapport de PRD de BERVET Maël et étude de l'heuristique dédiée (version Bervet).
29/09/2021	Lecture des documents concernant l'heuristique dédiée communiqués par V. T'kindt. Etude de l'heuristique dédiée (version Bervet).
30/09/2021	Réunion avec R. Bocquillon : premier déroulement et explication de l'heuristique dédiée, questionnements sur quelques points à revoir. Retravail de l'heuristique dédiée et résolution des questionnements. Nouvelles propositions à confirmer.
06/10/2021	Intervention de J.Y. Ramel et N. Ragot : importance de l'état de l'art et des spécifications. Réécriture du déroulement de l'heuristique (partie construction du graphe) et test de celle-ci sur un problème-exemple.
07/10/2021	Revue de l'heuristique et corrections. Réunion avec R. Bocquillon : revue de la construction du graphe de l'heuristique dédiée. Il reste quelques points à revoir. Fin de la réécriture du déroulement de l'heuristique dédiée (partie construction du graphe). Rendu document à R. Bocquillon et V. T'kindt : heuristique_dediee_explications_v2.
13/10/2021	Relecture et finitions de l'explication du fonctionnement de l'heuristique dédiée (partie construction du graphe). Ajout d'un exemple problème et déroulement de l'heuristique (partie graphe) sur ce problème.
14/10/2021	Rendu document à R. Bocquillon et V. T'kindt : heuristique_dediee_explications_v3.pdf, heuristique_dediee_probleme_exemple_v1.pdf, heuristique_dediee_deroule_exemple_v1.pdf, resume_hebdomadaire_v1.pdf.
20/10/2021	Ajout de la partie "arbre de recherche" dans le document d'explication de l'heuristique dédiée et dans le déroulé du problème exemple. Rendu document à R. Bocquillon et V. T'kindt : heuristique_dediee_explications_v4.pdf, heuristique_dediee_probleme_exemple_v2.pdf, heuristique_dediee_deroule_exemple_v3.pdf.
21/10/2021	Retravail sur l'heuristique et gestion de projet.
27/10/2021	Début des spécifications pour le rapport de PRD. Réunion avec J.Y. Ramel : première vue d'ensemble sur les spécifications et conseils.
28/10/2021	Suite des spécifications pour le rapport de PRD. Gestion de projet et suite de la prise en main de l'heuristique.
10/11/2021	Intervention de M. Hoguet et A. Vallein : présentation des réunions de suivi de projet. Suite des spécifications pour le rapport de PRD.
17/11/2021	État de l'art et suite des spécifications pour le rapport de PRD.
18/11/2021	Réunion avec R. Bocquillon et V. T'kindt : présentation de l'avancement du projet, revue de l'heuristique et discussion sur la suite
24/11/2021	État de l'art et Gantt. Réflexion sur l'heuristique suite à la réunion de la semaine précédente.
25/11/2021	Gestion de projet et Gantt. Rendez-vous avec Amaury VALLEIN : première réunion concernant le suivi de projet. Suite de réflexion sur l'heuristique et questionnements. État de l'art. Rendu document à R. Bocquillon et V. T'kindt : compte-rendu_reunion_20211118.pdf, resume_hebdomadaire_v2.pdf.
01/12/2021	Rédaction de l'état de l'art.

Figure A.4 – Suivi de tâches - partie 1

02/12/2021	Rédaction de l'état de l'art et du rapport.
08/12/2021	Rédaction du rapport.
09/12/2021	Rédaction du rapport.
15/12/2021	Rédaction du rapport et étude du code de M. Bervet.
16/12/2021	Rédaction du rapport et étude du code de M. Bervet.
05/12/2022	Etude du code de M. Bervet.
06/01/2022	Etude du code de M. Bervet.
12/01/2022	Etude du code de M. Bervet.
13/01/2022	Rendez-vous avec V. T'kindt : point sur l'avancement du projet, planning pour la suite et éléments importants. Etude du code de M. Bervet.
19/01/2022	/
20/01/2022	/
26/01/2022	/
27/01/2022	/
02/02/2022	/
03/02/2022	/
09/02/2022	/
10/02/2022	/
23/02/2022	/
24/02/2022	/
02/03/2022	Rendez-vous avec V. T'kindt : point sur les problèmes qui n'ont pas permis l'avancement prévue du projet et reciblage des attendus. Gestion de projet et Gantt.
03/03/2022	Etude du code de M. Bervet et du générateur d'instance.
09/03/2022	Etude du code de M. Bervet et du générateur d'instance.
10/03/2022	Etude du code de M. Bervet et du générateur d'instance. Gestion de projet et Gantt. Rédaction du rapport de suivi de projet.
16/03/2022	Etude du code de M. Bervet et du générateur d'instance. Développement du générateur d'instances.
17/03/2022	Etude du code de M. Bervet et du générateur d'instance. Développement du générateur d'instances.
23/03/2022	Rendez-vous avec R. Bocquillon et V. T'kindt : point sur l'avancée. Développement du générateur d'instances.
24/03/2022	Développement du générateur d'instances.
30/03/2022	Développement du générateur d'instances. Réalisation de quelques tests.
31/03/2022	Rendez-vous avec R. Bocquillon et V. T'kindt : point sur les rendus. Rédaction du rapport.
06/04/2022	/
07/04/2022	/

Figure A.5 – Suivi de tâches - partie 2

# B

## Description des interfaces

### 1 Interfaces matérielles/logicielles

La partie développée pendant ce projet communiquera seulement avec le reste du logiciel d'ordonnancement. Il recevra les données sous forme de classes et retournera une solution fonctionnelle.

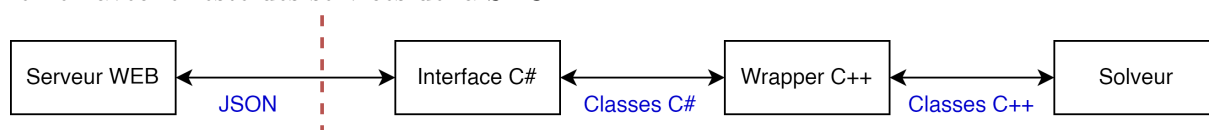
Le logiciel d'ordonnancement, duquel nous allons donc développer le noyau de recherche opérationnelle, communique cependant quant à lui avec les serveurs de la SNCF par fichier JSON afin de recevoir les instances et de renvoyer les solutions. Mais cela est une partie sur laquelle nous n'allons pas travaillé puisqu'elle est déjà fonctionnelle. Notre objectif ici sera juste de remplacer le noyau de calcul (le solveur) par l'implémentation de l'heuristique.

### 2 Interfaces homme/machine

Aucune interface homme/machine n'est à prévoir pour ce projet. Le logiciel d'ordonnancement ne communique qu'avec les services de la SNCF qui, de leur côté, font le travail d'interfaçage avec les opérateurs.

### 3 Interfaces logicielles/logicielles

La partie que nous allons développer dans ce projet communique avec le reste du logiciel d'ordonnancement. Elle récupère les données via des classes et retransmet le résultat sous le même format. C'est le reste du logiciel (dont nous ne nous occupons pas) qui se charge de faire le lien avec le reste des services de la SNCF.



**Figure B.1** – Schéma présentant l'organisation du logiciel d'ordonnancement

Le schéma ci-dessus présente les interfaces entre les différentes parties du logiciel d'aide à la décision. Lors de ce projet, nous développerons que la partie solveur.



# Cahier de Spécifications

## 1 Spécifications fonctionnelles

### 1.1 Générateur d'instance

#### Générer une instance

##### INSaddInstance

Entrée :

- Nombre de centres
- Nombre de voies
- Nombre d'opérations

Sortie : Instance

Préconditions : none

Postconditions : none

#### Afficher les instances

##### INStoString

Entrée : (Instances)

Sortie : Texte représentant les instances

Préconditions : none

Postconditions : none

#### Générer un fichier JSON

##### INStoJSON

Entrée : (Instances)

Sortie : Instances sous le format JSON

Préconditions : none

Postconditions : none

## 2 Spécifications non fonctionnelles

### 2.1 Contraintes de développement et conception

La plus grosse contrainte de développement et de conception sera de reprendre le début d'implémentation déjà effectué. En effet, cela va demander du temps d'adaptation et de prise en main avant de pouvoir compléter l'implémentation et la déboguer. Il faudra aussi faire attention au format des données (reçu par le programme sous format JSON puis convertit en classes pour que nous puissions nous en servir dans notre partie).

L'implémentation existante a été réalisée en C++ dans un projet Visual Studio. Ce sont deux éléments qui doivent être pris en compte lors de la reprise du projet existant. Il faudra se conformer à ces contraintes mais aussi prendre en main les librairies utilisées dans cette première implémentation.

### 2.2 Contraintes de fonctionnement et d'exploitation

#### 2.2.1 Performances

Le programme n'a pas besoin d'avoir des performances du niveau d'un programme temps réel. Cependant, puisqu'il est question d'ordonnancement, il va falloir faire attention à ce que notre application puisse donner une solution dans un délais convenable.

#### 2.2.2 Capacités

Le programme tournera sur un serveur de la SNCF. La capacité de mémoire dépend donc seulement du serveur sur lequel il tournera. On considère que cette contrainte n'est pas très forte dans les mesures du raisonnable.

#### 2.2.3 Contrôlabilité

En fonction de ce qui a été fait dans le début d'implémentation, un fichier de logs pourra permettre de suivre l'exécution du programme (en développant les résultats intermédiaires de l'heuristique pour la solution proposée).

#### 2.2.4 Sécurité

La partie développée dans ce projet ne sera pas accessible directement donc pas de contrainte de ce côté là non plus. Elle fonctionnera seulement en recevant les données d'instance et renverra les solutions.

# D

# Cahier du développeur

## 1 Introduction

Ce cahier du développeur détaillera la partie "réalisation" effectuée pendant le semestre 10, second semestre du PRD. Cette partie couvre l'extraction et le développement du générateur d'instances dans un projet séparé du projet VPLS, intégré par Valentine Jourdan.

Ce cahier explicitera donc le développement et la modélisation du générateur d'instances.

## 2 Architecture du projet

Le projet se découpe en trois grandes parties distinctes :

- les **librairies**
- la classe **Instances**
- le fichier main (**InstanceGenerator.cpp**)

### 2.1 Les librairies

Une première grande partie se compose des différentes librairies. On y retrouve deux librairies utilisées pour simplifier la programmation et une "librairie" permettant de créer des instances.

La première librairie est la librairie **effolkronium/random** en version 1.4.0. Elle permet de générer des nombres aléatoires ou de gérer l'aléatoire en général de façon simple. Cette librairie était déjà utilisée dans le code des classes extraites du projet VPLS, c'est pourquoi nous nous sommes permis de l'utiliser ici aussi.

La seconde librairie importée est **nlohmann/json** en version 3.10.5. Elle permet de créer des objets JSON manipulables très simplement.

La dernière "librairie" est en réalité les classes extraites du projet VPLS, programmées par Valentine Jourdan. Elles sont intégrées dans le projet du générateur d'instance comme une sorte de librairie en "boîte noire" puisqu'il n'y avait aucun intérêt à y toucher.

## 2.2 La classe Instances

La seconde grande partie est la classe **Instances**. Cette classe permet de gérer les instances. Plus de détails sont disponibles dans la partie consacrée aux descriptions plus détaillées

## 2.3 InstanceGenerator.cpp

La dernière partie ne comprend que le fichier **InstanceGenerator.cpp**. Ce fichier fait office de fichier main. C'est lui qui récupère les différentes paramètres entrés et appelle les méthodes des différentes classes nécessaires pour générer les instances et les enregistrer dans le (ou les) fichier(s) JSON correspondant. Plus de détails sont disponibles dans la partie consacrée aux descriptions plus détaillées

# 3 Descriptions détaillées de données exploitées

Le générateur d'instance traite deux types de données différentes :

- les **instances** qu'il génère
- les **JSON**

## 3.1 Les instances

Les instances sont représentées dans le programme par une classe **InstanceData** qui contient tout les données d'une instance : le nombre de trains, de voies, d'infrastructures, ... mais aussi les arrêts, les sites de maintenance, les creux de roulement, ...

Une instance est générée grâce à la méthode *createAndFormat()* de la classe *InstanceDataFormatter*. Cette méthode retourne un objet *InstanceData* généré aléatoirement grâce au paramètres indiqués en entrée.

<i>InstanceDataFormatter</i>
- ... : ...
+ createAndFormat(double freq6, double freq5, double freq4, double freqVoies, std::vector freqInfraCentre) : InstanceData

**Figure D.1** – Diagramme de la classe *InstanceDataFormatter*

La plus part des données sont enregistrées dans les attributs de l'objet *InstanceData* créé sous forme de vecteurs de maps donc les clés sont des chaînes de caractères et les valeurs des entiers. Il est donc aisé d'accéder à une donnée si l'on connaît sa clé.

.

<i>InstanceData</i>
<ul style="list-style-type: none"> <li>- _dateLimit : unsigned int</li> <li>- _nbTrains : unsigned int</li> <li>- _nbTracks : unsigned int</li> <li>- _nbMaintSites : unsigned int</li> <li>- _nbTypesInfras : unsigned int</li> <li>- _nbTimeShifts : unsigned int</li> <li>- _nbTotalMaxStoppedTrainsTimeSlots : unsigned int</li> <li>- _nbTrainsStops : unsigned int</li> <li>- _begin : unsigned int</li> <li>- _trainsData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _tracksData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _maintSitesData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _typesInfrasData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _timeShiftsData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _totalMaxStoppedTrainsPerTimeSlotData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _trainsStopsData : vector&lt;map&lt;string, unsigned int&gt;&gt;</li> <li>- _corrMaxStoppedTrainsPerDate : vector&lt;unsigned int&gt;</li> <li>- _trainsNeededInfras : vector&lt;vector&lt;bool&gt;&gt;</li> <li>- _tracksEquippedInfras : vector&lt;vector&lt;bool&gt;&gt;</li> <li>- _maintSitesReachableFromTrainsStops : vector&lt;vector&lt;float&gt;&gt;</li> <li>- _trainsSchedule : vector&lt;vector&lt;map&lt;string, unsigned int&gt;&gt;&gt;</li> <li>- _tracksSchedule : vector&lt;vector&lt;map&lt;string, unsigned int&gt;&gt;&gt;</li> <li>- _maintSiteMaxInputPerTimeShift : vector&lt;vector&lt;unsigned int&gt;&gt;</li> </ul>
+ ... : ...

Figure D.2 – Diagramme de la classe *InstanceData*

## 3.2 Les JSON

Les JSON sont des objets créés grâce à la librairie **nlohmann/json** et sont très facilement manipulables. Le format du fichier JSON généré à partir des instances doit correspondre à ce qui est attendu en entrée de la partie "résolution du problème d'affectation". Ce format est explicité sous forme d'arbre, à la fin du chapitre.

## 4 Descriptions détaillées des classes, modules, réalisations

### 4.1 La classe Instances

La classe **Instances** a été développée pour permettre de gérer les instances plus facilement et notamment les objets `InstanceData`. Cette classe comprend 4 attributs :

- **instancesDatas** : un vecteur d'instances (ou plutôt d'objets "InstanceData")
- **criticitiesDatas**, **infrastructuresDatas** et **infrastructuresFrequencies** : des données fixes quel que soit l'initialisation de l'objet permettant de générer des instances cohérentes

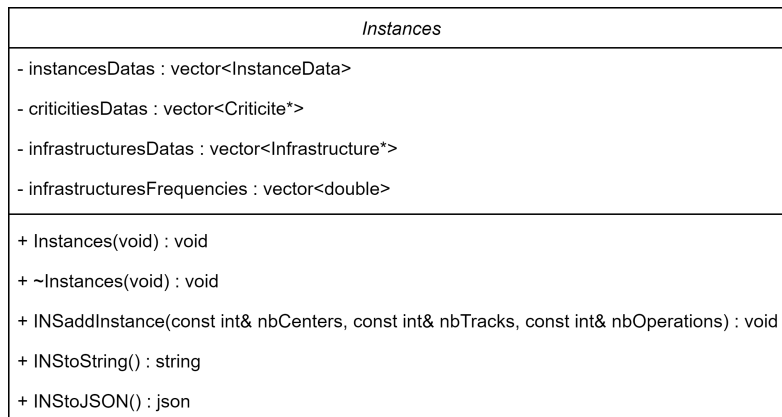


Figure D.3 – Diagramme de la classe Instances

Cette classe comporte aussi plusieurs méthodes. Parmi celles-ci on retrouve évidemment un constructeur par défaut et un destructeur permettant notamment d'initialiser et de détruire les attributs paramétrant la génération des instances.

Les trois dernières méthodes représentent enfin la partie la plus importante de la classe Instances. La méthode **INSaddInstance** permet de générer aléatoirement une instance (un objet `InstanceData`) avec x sites de maintenance, y voies par site et z opération à réaliser ; x, y et z étant entrés en paramètres. Pour cela, la méthode utilise la méthode `createAndFormat()` de la classe `InstanceDataFormatter`. Cette instance est alors ajoutée au vecteur des instances de la classe Instances, l'attribut `instancesDatas`.

La méthode **INStoString** permet de retourner une chaîne de caractères comportant toutes les données de toutes les instances présentes dans le vecteur d'instances.

La méthode **INStoJSON** permet de retourner un objet JSON (de la librairie `nlohmann/json`) au bon format lisible et interprétable par le projet qui résout le problème d'affectation et contenant toutes les instances.

### 4.2 Le fichier main : InstanceGenerator.cpp

Le fichier **InstanceGenerator.cpp** fait office de fichier main. Il récupère les paramètres entrés par l'utilisateur et crée les instances en fonction avant de les enregistrer dans le (ou les) fichier(s) JSON souhaités. Pour cela, il crée une instantiation de la classe Instances et, pour les différents paramètres entrés, itère la méthode `INSaddInstance`. A la fin, il ne lui reste plus qu'à utiliser la méthode `INStoJSON` et enregistrer l'objet JSON obtenu dans le (ou les) fichier(s) qui convient.

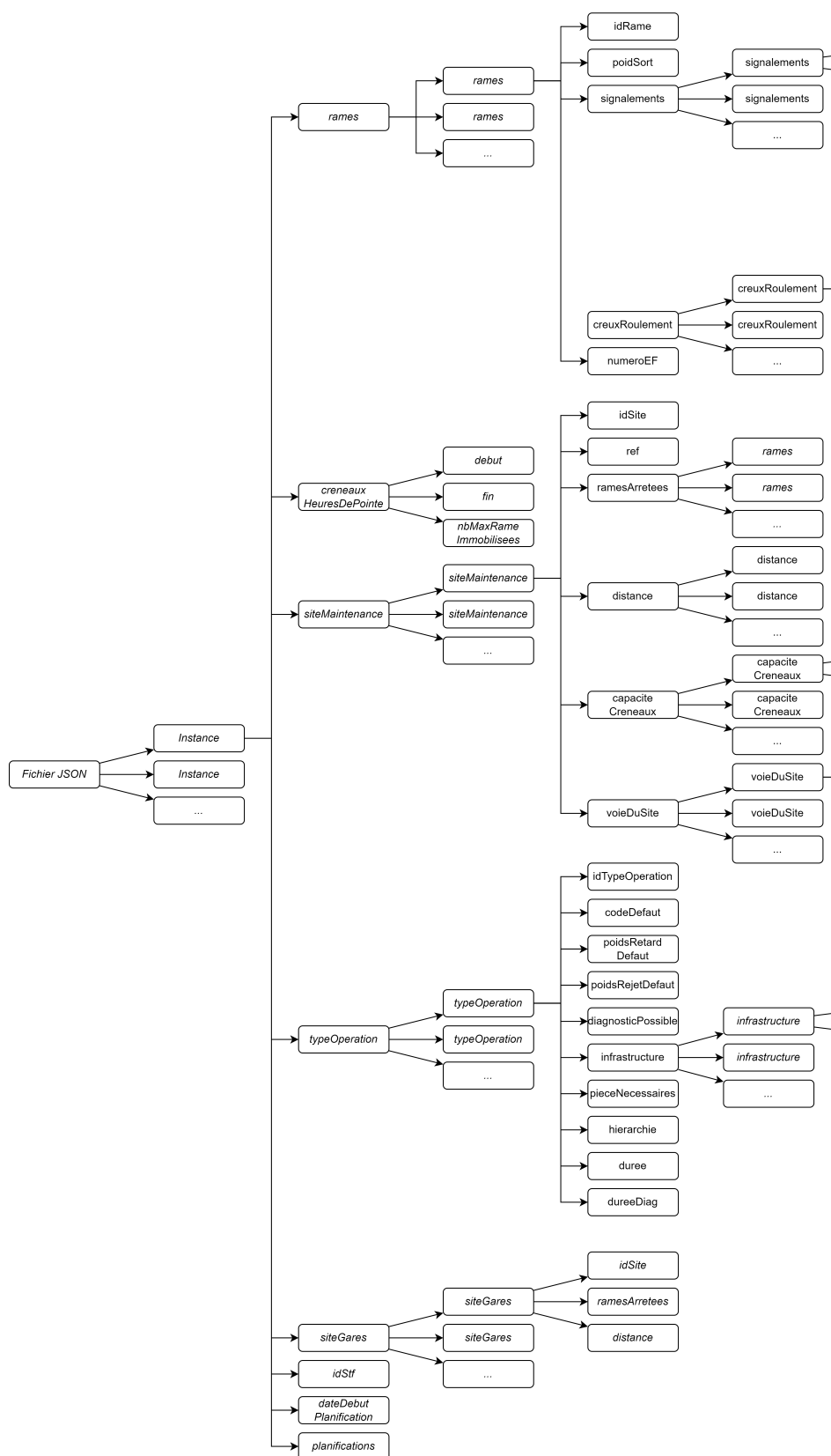


Figure D.4 – Diagramme du format JSON d'instance - partie 1

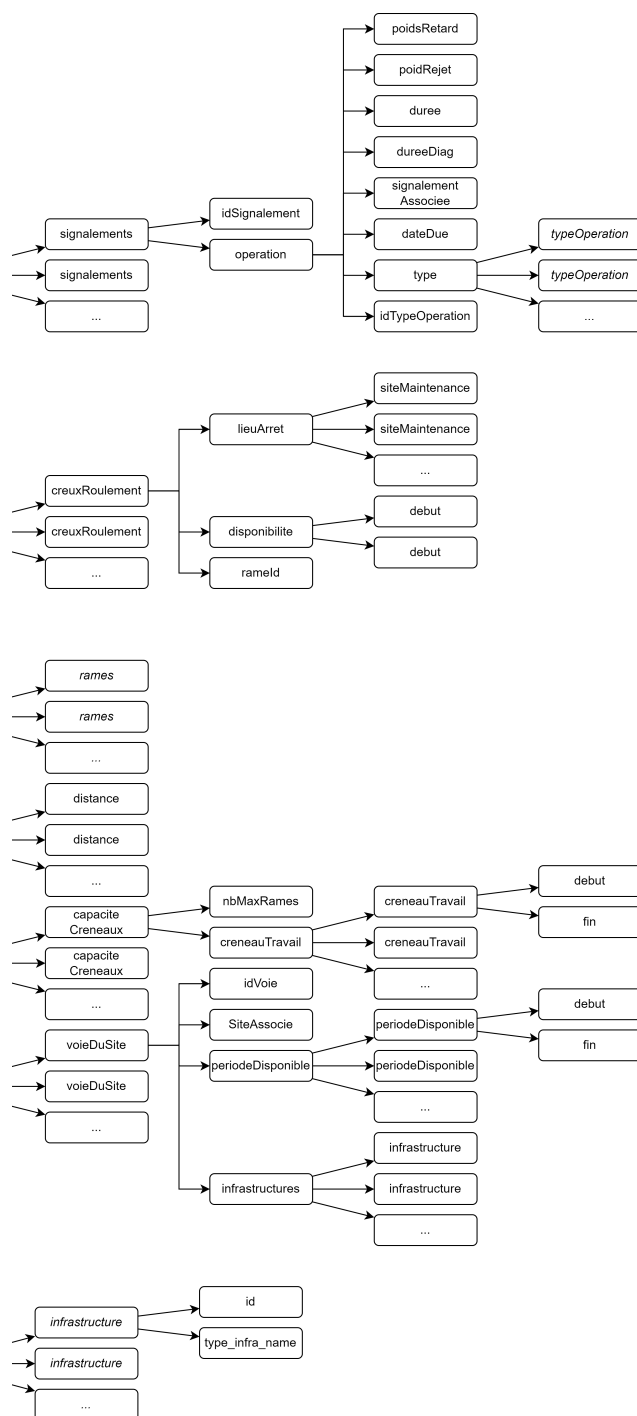


Figure D.5 – Diagramme du format JSON d'instance - partie 2

# E

## Document d'installation

Pour installer le générateur d'instances, rien de plus simple. Il suffit de disposer d'un ordinateur fonctionnant avec un système d'exploitation Windows et d'avoir préalablement récupéré l'archive des livrables de ce projet.

Pour utiliser l'application, il n'y a rien de plus à faire. Le fichier exécutable *InstanceGenerator.exe* est prêt à l'utilisation (plus de détails dans le [document d'utilisation](#)).

Pour récupérer le projet, celui-ci est disponible dans le dossier *InstanceGenerator*. Il s'agit d'un projet Visual Studio 2019. S'y trouve aussi un dossier doxygen avec la documentation nécessaire. Les librairies utilisées y sont sauvegardées dans le dossier *libs*.

Enfin, un dossier *Documents* regroupe tous les documents qui permettent de continuer le projet.

Les librairies sont intégrées directement dans le projet. Les deux seules utilisées sont :

- la librairie **random** (<https://github.com/effolkronium/random>)
- la librairie **json** (<https://github.com/nlohmann/json>)

# F

## Document d'utilisation

Le générateur d'instance s'utilise simplement en ligne de commande à condition d'y accoler les bons paramètres :

**InstanceGenerator.exe** <fichier\_entree> <fichier\_sortie> <multi-JSON>

Les paramètres à indiquer :

- <fichier\_entree> est à remplacer par le nom du fichier de paramètres. Si le fichier n'existe pas, un fichier portant ce nom sera créé avec des paramètres par défaut. Il est aussi possible d'indiquer "" pour utiliser les paramètres par défaut sans créer de fichier de paramètres.
- <fichier\_sortie> doit être substitué par le nom du fichier de sortie sous le format <nom>.json
- <multi-JSON> indique l'emplacement ou doit être indiqué si l'on souhaite séparer les différentes instances générées en plusieurs fichiers JSON (1) ou enregistrer toutes les instances dans un même fichier JSON (0).



# Cahier de test

NOM DU TEST
Test des paramètres
DESCRIPTION DU TEST
<p>L'objectif de ce test est de lancer le programme .EXE du générateur d'instance et de vérifier si pour toutes les possibilités des différents paramètres, le programme répond comme souhaité.</p> <p>Cas 1 : On exécute <i>InstanceGenerator.exe "" instance.json 0</i></p> <p>Cas 2 : On exécute <i>InstanceGenerator.exe instanceParam.txt instance.json 0</i> &amp; pas de fichier "instanceParam.txt" existant</p> <p>Cas 3 : On exécute <i>InstanceGenerator.exe instanceParam.txt instance.json 1</i> &amp; un fichier "instanceParam.txt" existant</p>
RÉSULTAT ATTENDU
<p>Cas 1 : Génération des instances dans un fichier "instance.json" à partir des paramètres par défaut</p> <p>Cas 2 : Création d'un fichier de paramètres comprenant les paramètres par défaut et génération des instances dans un fichier "instance.json"</p> <p>Cas 3 : Récupération des données du fichier d'entrée et génération des différentes instances dans des fichiers "instanceX.json" différents</p>
RÉSULTAT OBTENU
<p>Cas 1 : Génération des instances dans un fichier "instance.json" à partir des paramètres par défaut</p> <p>Cas 2 : Création d'un fichier de paramètres comprenant les paramètres par défaut et génération des instances dans un fichier "instance.json"</p> <p>Cas 3 : Récupération des données du fichier d'entrée et génération des différentes instances dans des fichiers "instanceX.json" différents</p>

NOM DU TEST
Test de conformité du JSON
DESCRIPTION DU TEST
L'objectif de ce test est de générer plusieurs instances JSON aléatoirement et de les fournir au solveur déjà en place pour s'assurer que le format JSON est bien correct et lu par le logiciel de résolution.
RÉSULTAT ATTENDU
Le fichier JSON est correctement lu et interprété. Aucune erreur n'est retourné.
RÉSULTAT OBTENU
Le fichier JSON est correctement lu et interprété. Aucune erreur n'est retourné.

H

Explication détaillée de l'heuristique dédiée

# Explication heuristique dédiée

jeudi 9 décembre 2021 14:00

Par Maël Bervet

Avec l'aide de Vincent T'kindt, Ronan Bocquillon et Valentine Jourdan

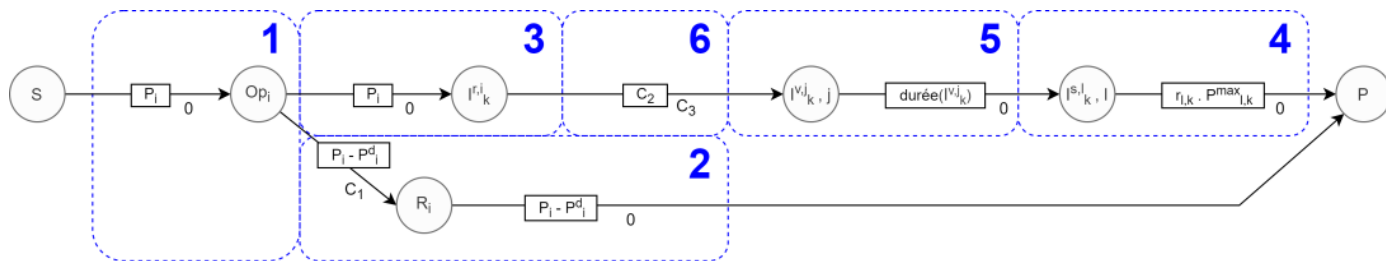
Repris et rédigé par Vincent Le Garjan.

Ce document présente une idée générale du fonctionnement de l'heuristique dédiée.

## Construction du graphe de flot maximum à coût minimum

Notre objectif sera de créer un graphe de flot maximum à coût minimum qui permettra de résoudre notre problème d'affectation. Deux sommets "source" et "puits" sont placés aux extrémités du graphe. Le flot traversant le graphe entre les sommets "source" et "puits" représentera la quantité de temps passé sur les différentes opérations.

**/!** Cette heuristique ne prend pas encore en compte les croisements. Aussi, cet heuristiques ne prend pas en compte certaines possibilités. Ces deux points feront l'objet d'une amélioration de l'heuristique un peu plus tard.



Quelques notations utiles concernant les intervalles de disponibilité :

- $I_k^{r,i}$  le k-ième intervalle de disponibilité de la rame  $i$
- $I_k^{v,j}$  le k-ième intervalle de disponibilité de la voie  $j$
- $I_k^{s,l}$  le k-ième créneau du site  $l$

Quelques notations supplémentaires pour simplifier les écritures :

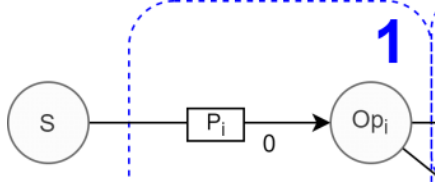
- $debut(I)$  : date de debut de l'intervalle  $I$
- $fin(I)$  : date de fin de l'intervalle  $I$
- $durée(I) = fin(I) - debut(I)$  : durée de l'intervalle  $I$

Les autres notations :

- $P_i$  la durée de l'opération  $i$
- $P_i^d$  la durée de diagnostic pour l'opération  $i$
- $d_i$  la date de rentrée souhaitée de la rame  $i$  (ou "date due")
- $D$  l'horizon de planification (la date la plus tardive de la planification)
- $w_i$  le poids de retard de la rame  $i$  (ou "priorité de la rame")
- $u_i = w_i \times (D - d_i)$  le poids de rejet de la rame  $i$
- $r_{l,k}$  le nombre maximum de rentrées de rames sur le site  $l$  pendant le créneau  $k$
- $P_{l,k}^{max} = \max(P_{i \in E})$  avec  $E$  : l'ensemble des rames pouvant être réparées sur l'une des voies du site  $l$ , pendant le créneau  $k$

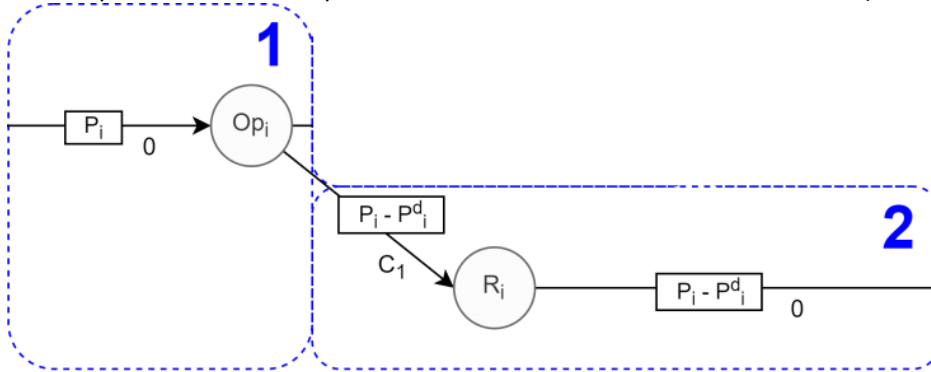
## 1 - Opération

On commence par créer un sommet  $Op_i$  par opération que l'on souhaite réaliser et on le relie à la source par un arc de capacité  $P_i$  et de coût 0.



## 2 - Rejet

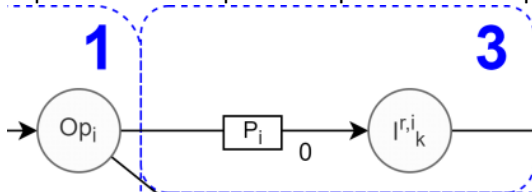
Pour chaque sommet "opération", on crée un sommet "rejet"  $R_i$ . Ce sommet nous permet de prendre en compte le cas où la rame ne subirait qu'un diagnostic seulement et où l'opération serait "rejetée" à plus tard (à une date plus tardive que notre horizon de planification). Ce sommet est relié d'un côté au sommet "opération" par un arc de capacité  $P_i - P_i^d$  et de coût  $C_1$  (voir juste après). Il est aussi relié de l'autre côté au puit par un arc de capacité  $P_i - P_i^d$  (nous avons laissé cette capacité mais celle-ci pourrait être enlevée ou définie comme infinie) et de coût 0.



Selon la première version de l'heuristique,  $C_1$  valait  $u_i / P_i$ . Nous proposons une nouvelle version dans laquelle  $C_1$  vaudrait  $u_i / (P_i - P_i^d)$ .

## 3 - Intervalles rame

On ajoute ensuite un sommet par intervalle de disponibilité de la rame devant subir l'opération. Nous ne prenons en compte que les intervalles qui sont au moins assez grand pour faire passer l'opération en diagnostique. On relie les sommets de disponibilité des rames aux sommets "opération" correspondants par un arc de capacité  $P_i$  et de coût 0.



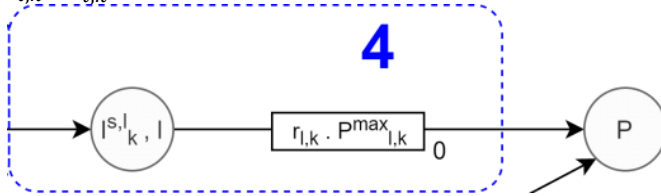
Si un intervalle de disponibilité d'une rame recouvre la date due de l'opération (c'est-à-dire si la date de rentrée souhaitée de la rame se situe dans un intervalle de disponibilité de la rame), le sommet correspondant est divisé en deux sommets distincts, l'un ( $I_{k,1}^{r,i}$ ) se terminant à la date due, l'autre ( $I_{k,2}^{r,i}$ ) commençant à cette même date. L'arc reliant le premier sommet au sommet "opération" gardera la capacité initiale ( $P_i$ ) alors que l'arc reliant le second sommet au sommet "opération" aura une capacité égale à la durée de cet intervalle ( $durée(I_{k,2}^{r,i})$ ). Si cette dernière capacité est trop petite pour faire passer l'opération en diagnostique ( $durée(I_{k,2}^{r,i}) < P_i^d$ ), le sommet est supprimé.

## 4 - Créneaux sites

À partir de cette étape, nous continuerons la construction de notre graphe en partant de la fin (du puit).

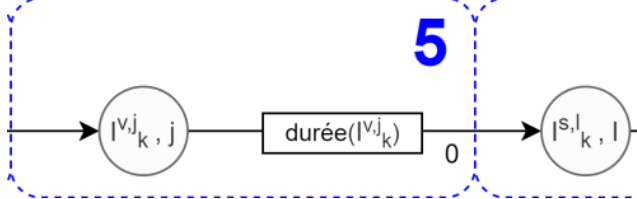
On crée un sommet par créneau des sites de maintenance (sur chaque sommet est aussi précisé le nom du site afin de le garder en mémoire). On relie ces sommets au puit par un arc de capacité

$r_{l,k} \cdot P_{l,k}^{max}$  et de coût 0.



## 5 - Intervalles voies

Ensuite, et pour chaque voie des sites de maintenance, on ajoute un sommet par intervalle de disponibilité de la voie (sur chaque sommet est aussi précisé le nom de la voie afin de le garder en mémoire). On relie ces sommets aux créneaux des sites correspondants par un arc de capacité  $durée(I_k^{v,j})$  et de coût 0. Un sommet "intervalle voie" est relié à un sommet "créneau site" si la date de début de l'intervalle "voie" appartient au créneau "site" (si  $debut(I_k^{v,j}) \in I_k^{s,l}$ ).

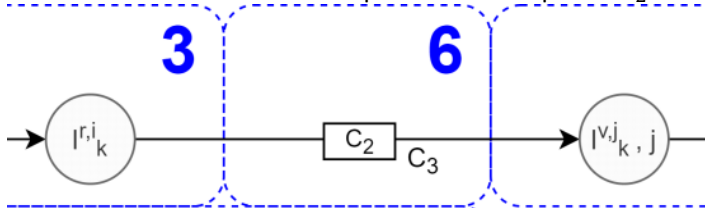


## 6 - Arcs rame-voie

Enfin, la dernière étape sera de relier les sommets "intervalle rame" aux sommets "intervalle voie". Deux sommets sont reliés si :

- l'intersection des intervalles permet de faire passer l'opération en diagnostique ( $durée(I_k^{r,i} \cap I_k^{v,j}) \geq P_i^d$ )
- la voie possède l'infrastructure permettant l'opération de la rame.

L'arc reliant les deux sommets possède une capacité  $C_2$  et un coût  $C_3$ .



Dans la première version de l'heuristique,  $C_2$  valait  $durée(I_k^{r,i})$ . Nous proposons d'utiliser plutôt  $durée(I_k^{r,i} \cap I_k^{v,j})$  comme valeur de capacité  $C_2$ . Lorsque l'intervalle rame a été découpé en deux sommets, on considérera que le premier sommet correspond à l'intervalle initial dans le calcul de la capacité des arcs sortants (on prendra  $I_k^{r,i}$  et non  $I_{k,1}^{r,i}$ ). Pour le second sommet, on prendra bien  $I_{k,2}^{r,i}$ .

$C_3$  vaut  $\max\left(w_i \times (debut(I_k^{r,i} \cap I_k^{v,j}) - d_i) / P_i, 0\right)$ .

## Prétraitement

Une fois le graphe construit, nous pouvons lui appliquer un prétraitement afin d'éliminer d'éventuels intervalles rames infaisables.

Pour cela, on passe la capacité des arcs reliant la source aux sommets "opération" à  $P_i^d$ . Puis, pour chaque sommet "intervalle rame" les uns après les autres, on va faire passer tout le flot correspondant au diagnostic de l'opération sur l'arc les reliant aux sommets "opération" (on met la capacité des autres arcs "opération - intervalles rames" à 0). Si le flot final obtenu est différent de la somme de tous les  $P_i^d$ , l'intervalle rame implique une infaisabilité et est supprimé du graphe.

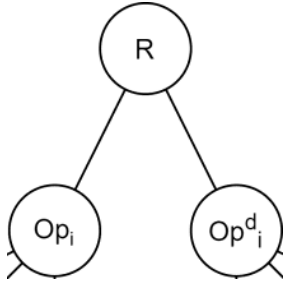
## Arbre de recherche

Dans la dernière partie de cette heuristique, nous allons construire l'arbre de recherche afin de trouver les meilleures affectations rame-voie et ainsi de résoudre notre problème. Cet arbre sera composé d'un nœud "racine" et de deux niveaux par opération à effectuer.

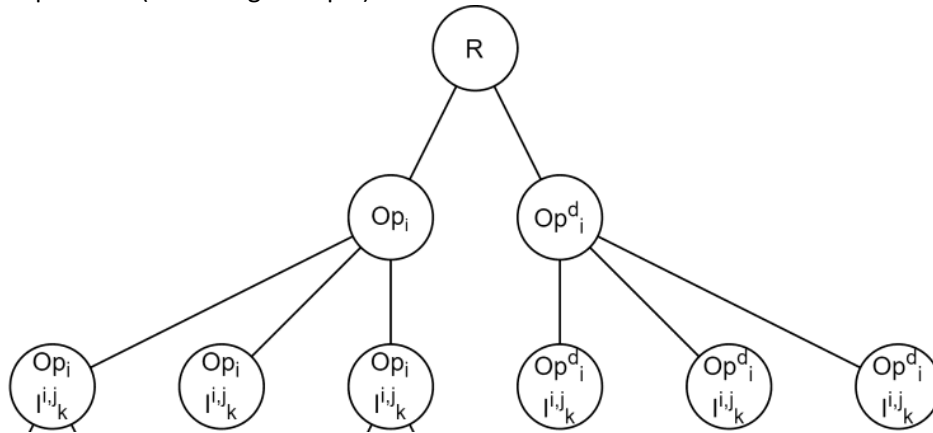
L'idée de cette partie de l'heuristique est de construire l'arbre au fur et à mesure, de façon dynamique, en suivant une démarche itérative (en répétant les étapes 1 et 2 jusqu'à avoir épuisé toutes les opérations à ordonnancer) et d'affecter un intervalle rame-voie à chaque opération. Nous créerons, en partant du nœud "racine" deux niveaux par opération en commençant par celle avec la criticité la plus élevée (si plusieurs opérations possèdent la même criticité, on sélectionnera d'abord celle avec la plus longue durée opératoire).

### 1 - Création d'un "double niveau"

On sélectionne la première opération (plus longue durée opératoire parmi la criticité la plus élevée) parmi celles restantes. On crée un premier niveau composé de 2 nœuds, permettant de choisir si on fait passer la totalité de la durée opératoire ou si on fait passer l'opération en diagnostique seulement ("rejet"). On relie les nœuds de ce niveau au nœud "racine" (si c'est le premier "double niveau") ou au nœud choisi du niveau supérieur.



On ajoute un second niveau en dessous du premier comportant, pour chacun des deux nœuds du niveau supérieur, un nœud par intervalle de disponibilité voie-rame ( $I_k^{r,i} \cap I_k^{v,j}$ ) pour la rame nécessitant l'opération. Ces nœuds permettront de choisir quel intervalle voie-rame utiliser pour l'opération (ou le diagnostique).

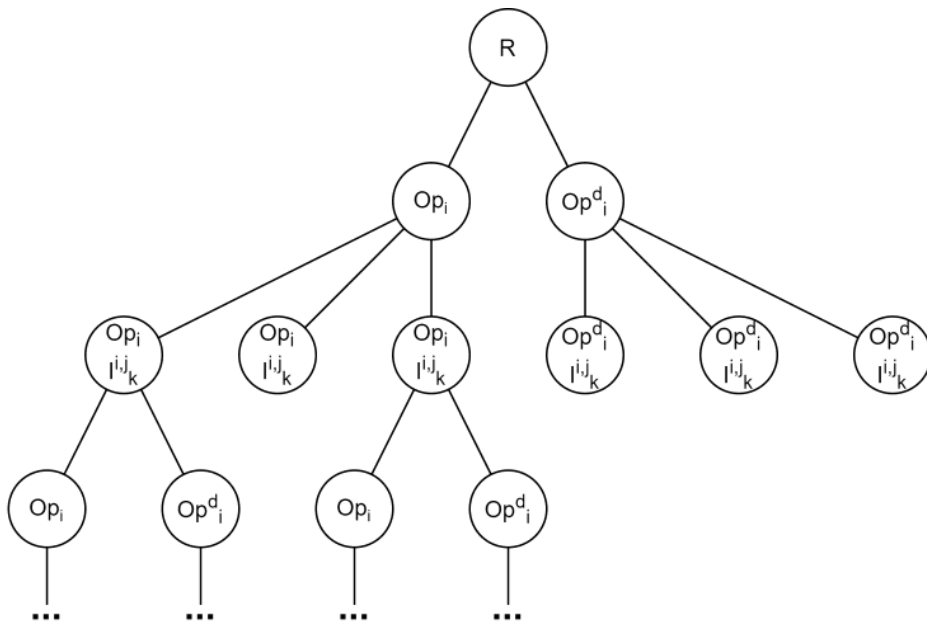


### 2 - Evaluation des nœuds

Ensuite, nous allons devoir évaluer les différents nœuds. Pour chaque nœud du niveau le plus bas, on met la capacité de l'arc rame-voie à  $P_i$  (ou  $P_i^d$  si on essaye de faire passer l'opération en diagnostique) et celle des autres à 0. On calcule le flot maximal et son coût. Parmi les nœuds faisables (flots sortant de la source et arrivant au puit identiques), on sélectionne les  $x$  nœuds les plus prometteurs (avec le coût le plus faible).

### 3 - Recommencer

On recommence les étapes (1) et (2) depuis chacun des nœuds sélectionnés en prenant en compte, évidemment, l'affectation qui vient d'être réalisée.



# I

## Exemple de fonctionnement de l'heuristique dédiée

# Problème exemple

jeudi 9 décembre 2021 14:00

Par Vincent Le Garjan

**Voici un exemple de problème simple afin de tester l'heuristique.**

Prenons une période de planification d'une journée (24h, allant des temps  $t=0$  à  $t=24$ ).

Notre système comporte :

- 2 rames

N° de la rame	Durée totale de l'opération / dont durée du diagnostique	Date due de l'opération	Infrastructure nécessaire	Disponibilités	Priorité
1	3 heures / 1 heure	$t=18$	Fosse	[00:04], [12:15], [19:24]	4
2	7 heures / 1 heure	$t=14$	Fosse + Pont	[00:20], [22:24]	6

- 1 site

N° du site	Créneaux	Nombre d'entrées possibles par créneau
1	[00:08], [08:16], [16:24]	1

- 3 voies

N° de la voie	Site correspondant	Infrastructure disponible	Disponibilités
1	1	Fosse + Pont	[07:09], [15:20]
2	1	Fosse	[06:12], [13:15], [20:24]
3	1	Pont	[07:10], [13:24]

# Déroulé de l'heuristique dédiée sur le problème exemple

jeudi 9 décembre 2021 14:00

Par Vincent Le Garjan

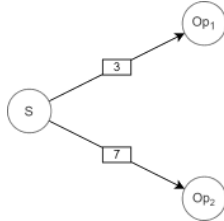
On va suivre les étapes données dans l'heuristique dédiée une à une.

## Construction du graphe de flot maximum à coût minimum

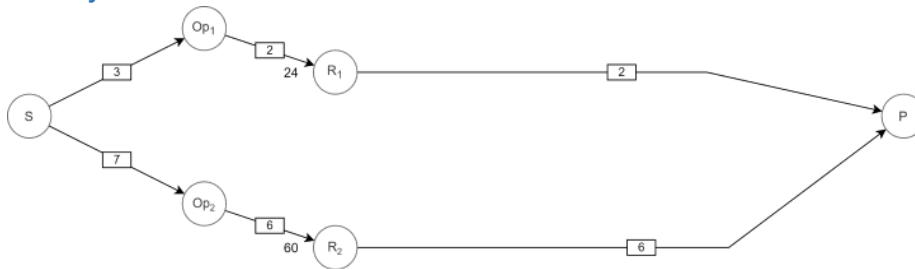
Pour simplifier la lisibilité, lorsqu'un arc possède un coût de 0, ce coût n'est plus indiqué.



### 1 - Opération

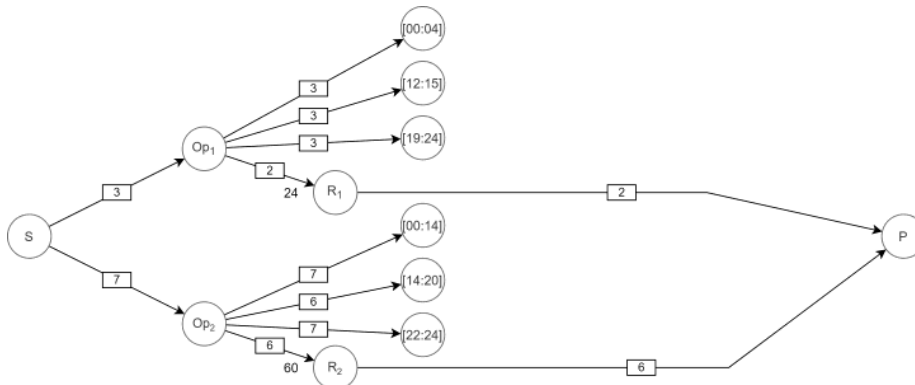


### 2 - Rejet

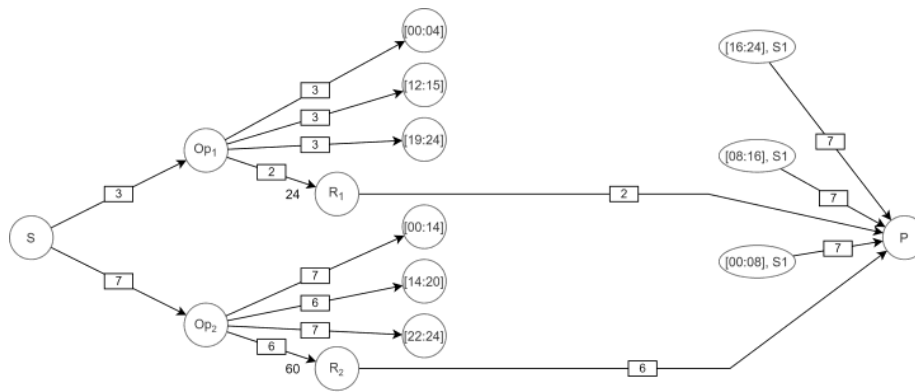


### 3 - Intervalles rame

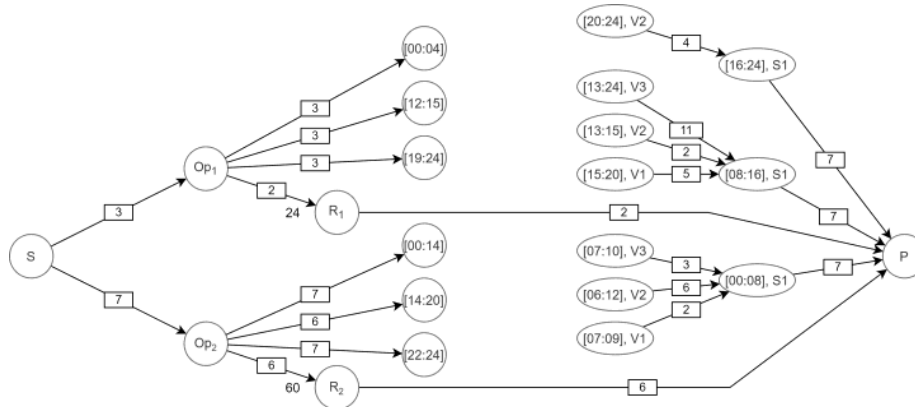
Note : on sépare l'intervalle de disponibilité [00:20] de la rame n°2 en deux sommets [00:14] et [14:20] car la "date due" de cette rame est à  $t = 14$ .



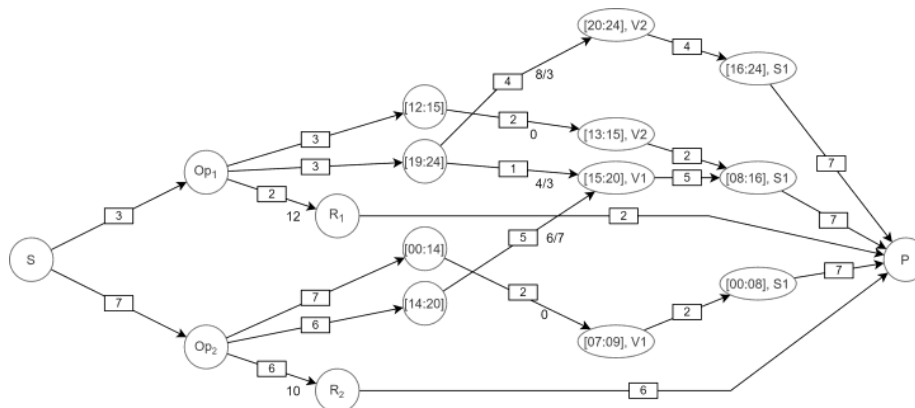
### 4 - Intervalles sites



## 5 - Intervalles voies



## 6 - Arcs rame-voie



## Prétraitement

Aucun sommet "intervalle rame" n'est supprimé suite à l'exécution du prétraitement.

## Arbre de recherche

Les nœuds colorés en **rouge** sont les nœuds "infaisables".

Les nœuds colorés en **bleu** sont les nœuds "faisables" non choisis.

Les nœuds colorés en **vert** sont les nœuds "faisables" choisis.

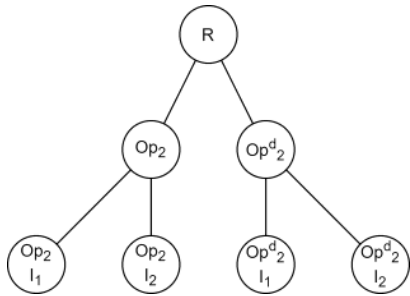
Les autres nœuds (en **blanc**) sont explorés de toutes façons.

Ici, on a décidé de prendre  $x = 1$  (le nombre de nœuds développés pour chaque opération).

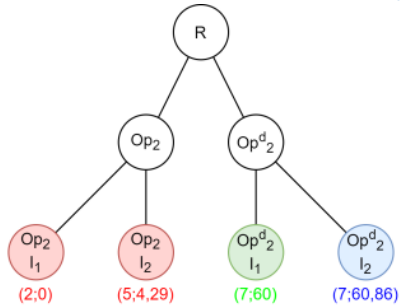
Pour plus de lisibilité, les intervalles voie-rame ont été nommés. Pour chaque rame, le nommage des intervalles commence par le premier (plus haut sur le graphe) intervalle "rame" et parmi ceux-ci, le premier intervalle "voie".

L'évaluation des nœuds se lit : (flot sur l'arc de l'intervalle rame-voie ; coût de la solution)

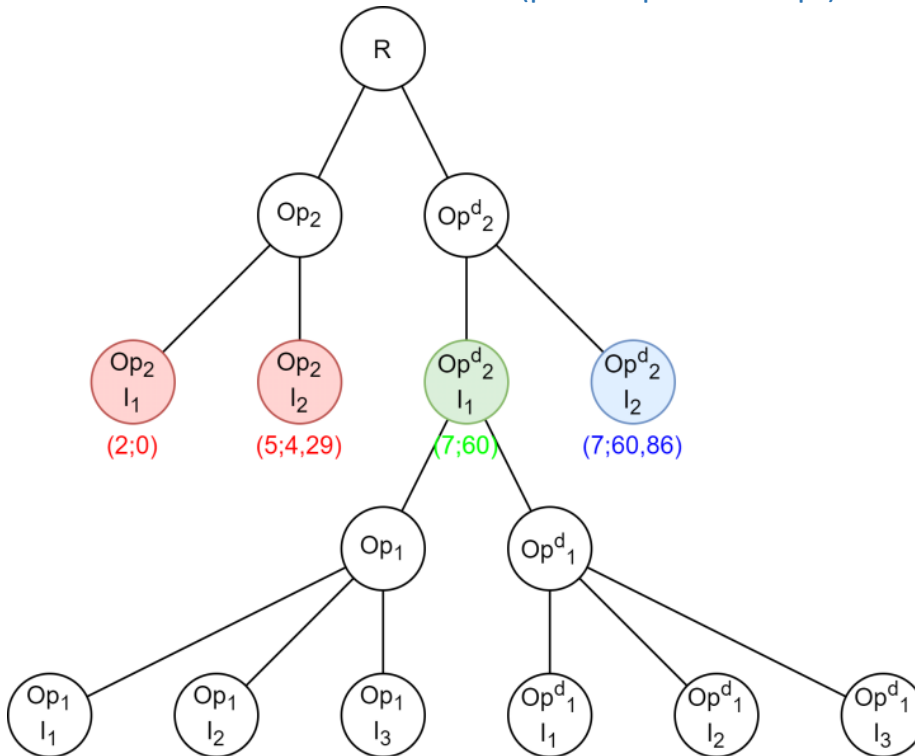
### 1a - Création d'un "double niveau" (pour l'opération $Op_1$ )



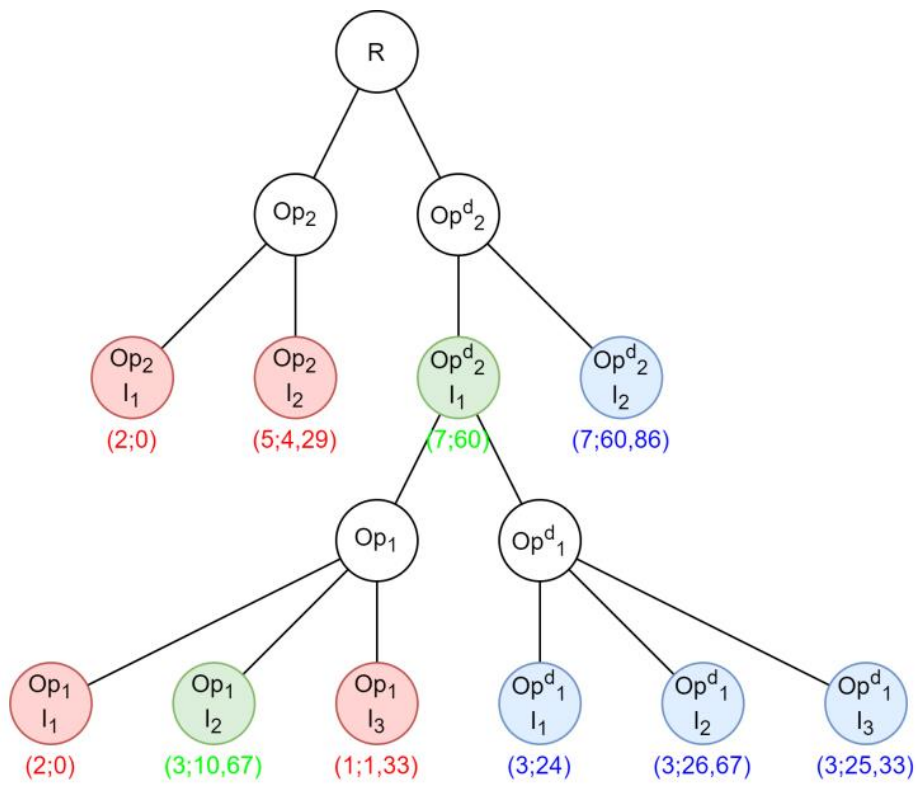
### 2a - Evaluation des nœuds (pour l'opération $Op_1$ )



### 1b - Création d'un "double niveau" (pour l'opération $Op_2$ )



### 2b - Evaluation des nœuds (pour l'opération $Op_2$ )



- [1] Maël BERVET. « Ordonnancement des opérations de maintenance corrective des trains au sein de la SNCF ». Projet Recherche & Développement. Tours, France : Ecole Polytechnique de l'Université François Rabelais de Tours, 2019-2020.
- [2] Valentine JOURDAN. « Ordonnancement de la maintenance corrective des lignes h/k du reseau transilien ». Master Optimisation et Recherche Opérationnelle. Tours, France : Université de Tours, 2019.
- [3] PERDEREAU, LORIS. *Modèle mathématique pour l'ordonnancement de la maintenance corrective des trains intégrant la possibilité de croisements*. Rapport de stage. Ecole Polytechnique de l'Université François Rabelais de Tours. Tours, France, 2021, p. 23-25.
- [4] Benjamin POUVREAU. « Ordonnancement des opérations de maintenance corrective des trains au sein de la SNCF ». Projet Recherche & Développement. Tours, France : Ecole Polytechnique de l'Université de Tours, 2018-2019.
- [5] Tom RAY. « Heuristique et programmation mathématique pour la planification de la maintenance corrective des trains au sein de la SNCF ». Projet Recherche & Développement. Tours, France : Ecole Polytechnique de l'Université de Tours, 2021-2022.
- [6] T'KINDT, VINCENT AND BOCQUILLON, RONAN AND BERVET, MAËL AND JOURDAN, VALENTINE. *Heuristique dédiée*. Document explicatif. Ecole Polytechnique de l'Université François Rabelais de Tours. Tours, France, 2019-2020.

# K

## Glossaire

**croisement** Un croisement à lieu lorsque deux rames, alors sur le même site en même temps, échangent leur planning de roulement. 2

**heuristique** Algorithme simple et/ou rapide permettant d'obtenir une solution approchée à un problème d'optimisation. 2

**ordonnancer** Disposer quelque chose dans un certain ordre, selon une certaine organisation. Attribuer une temporalité et une/des ressource.s (localité, ...) à des tâches. 1

**planning de roulement** Planning décrivant les périodes lors desquelles une rame est occupée par un voyage commercial ou de service. 2

**rames** files de wagons, ensembles de véhicules ferroviaires attelés entre eux. Trains avec leur ou leurs motrices. 1

# L

## Acronymes

**IDE** Integrated Development Environment. 3

**LIFAT** Laboratoire d'Informatique Fondamentale et Appliquée de Tours. 1

**MOA** Maitrise d'ouvrage. 2

**MOE** Maitrise d'œuvre. 2

**PLS** Programmation Linéaire Standard. 7

**PPC** Programmation Par Contraintes. 8

**PRD** Projet Recherche & Développement. 2

**SNCF** Société Nationale des Chemin de fer Français. 1



# Développement d'une heuristique dédiée pour la planification de la maintenance corrective des trains au sein de la SNCF

Vincent Le Garjan

Encadrement : Ronan Bocquillon et Vincent T'kindt

## Objectifs

L'objectif de ce projet est de continuer la recherche et l'implémentation d'une heuristique permettant l'ordonnement des maintenances correctives des rames des lignes H et K du Transilien. L'idée est donc de développer un algorithme permettant d'aider les agents de la SNCF à affecter une voie et un horaire de réparation à chaque train "malade", et ce dans un délais convenable, en fonction de la gravité de la panne.



Client : la SNCF - - Pôle ingénierie  
du matériel de Saint Pierre des Corps



En collaboration avec SNCF - Pôle ingénierie  
du matériel de Saint Pierre des Corps

## Résultats attendus

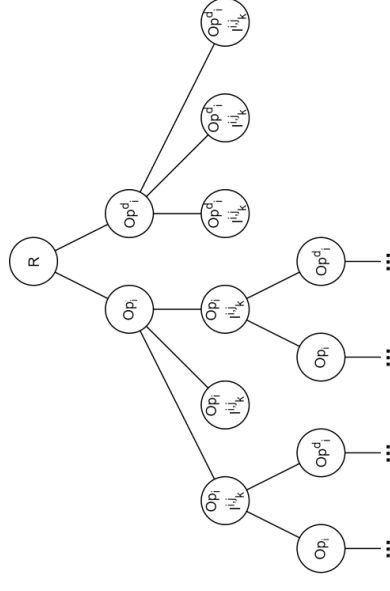
L'ambition de ce projet était de pouvoir récupérer, à la fin de l'exécution du programme, grâce à notre heuristique, une planification de maintenance des rames minimisant les retards en fonction de la gravité des pannes afin d'aider les agents de la SNCF à programmer le trajet des trains "malades". Mais le temps n'a permis que de réaliser le générateur d'instance des données d'entrée.

## Mise en œuvre

Afin de répondre à cet objectif, la première étape à réaliser a été de reprendre l'heuristique dédiée existante, de la prendre en main et de la compléter. Il s'agira ensuite de récupérer la première implémentation déjà effectuée de cette heuristique, de la corriger et de l'achever pour enfin l'intégrer dans le logiciel d'aide à la décision déjà développé.



Rame Z50000 (utilisée sur les  
lignes H et K du Transilien)



Arbre de recherche généré par l'heuristique pour l'affectation de rames à des voies



Ecole Polytechnique de l'Université de Tours  
Département Informatique  
64 avenue Jean Portalis, 37200 Tours, France  
[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

POLYTECH<sup>®</sup>  
TOURS

Informatique

# Développement d'une heuristique dédiée pour la planification de la maintenance corrective des trains au sein de la SNCF

## Résumé

Ce rapport détaille le travail réalisé par Vincent LE GARJAN sous la supervision de M. Ronan BOCQUILLON et M. Vincent T'KINDT au sein de Polytech TOURS et en partenariat avec la SNCF. Ce projet de recherche et développement (PRD) a été effectué dans le cadre de la cinquième année du département informatique de l'école d'ingénieur polytechnique de l'université de Tours.

L'objectif était de reprendre, améliorer et implémenter une heuristique dédiée permettant d'ordonnancer la maintenance corrective des trains au sein de la SNCF afin de fournir un logiciel d'aide à la décision pour les planificateurs. Ce projet s'inscrit dans le prolongement de trois PRD et d'un mémoire de master.

## Mots-clés

Polytech Tours, SNCF, Projet Recherche et Développement, Recherche opérationnelle, Heuristique, Ordonnancement, Planification, Aide à la décision, Maintenance corrective, Flot à coût minimum, Arbre de recherche en faisceau

## Abstract

This report details the work done by Vincent LE GARJAN under the supervision of Mr. Ronan BOCQUILLON et Mr. Vincent T'KINDT within Polytech Tours and in partnership with the SNCF. This research and development Project (RDP) was carried out as part of the fifth year of the Computer Department of the Polytechnic Engineering School of the University of Tours.

The objective was to take over, improve and implement a dedicated heuristic to schedule the corrective maintenance of trains within the SNCF in order to provide decision support system for planners. This project is an extension of three RDP and a master's thesis.

## Keywords

Polytech Tours, SNCF, Research and Development Project, Operations research, Heuristic, Scheduling, Planning, Decision support system, Corrective maintenance, Minimum-cost flow, Beam-search tree

## Entreprise

SNCF - Pôle ingénierie du matériel de Saint Pierre des Corps



## Tuteur entreprise

Romuald DETRUE

## Étudiant

Vincent LE GARJAN (DI5)

## Tuteurs académiques

Ronan BOCQUILLON

Vincent T'KINDT