



Ecole doctorale des sciences pour l'ingénieur

Doctorat
THÈSE

pour obtenir le grade de docteur délivré par

Université Clermont Auvergne

Spécialité doctorale “Informatique”

présentée et soutenue publiquement par

Eloise Yollande MOLÉ KAMGA

le Date soutenance

Pilotage synchronisé de la production d'énergie et d'activités de services de véhicules autonomes ou partiellement autonomes

Rapporteurs :

Examinateurs :

Directeurs de thèse :

Fatiha BENDALI,	Maîtresse de Conférences, HDR	Université Clermont Auvergne
Jean MAILFERT,	Maître de Conférences, HDR	Université Clermont Auvergne
Alain QUILLIOT,	Professeur des Universités	Université Clermont Auvergne
Hélène TOUSSAINT,	Ingénieure de Recherche CNRS	Université Clermont Auvergne

LIMOS

Laboratoire d'Informatique de Modélisation et d'Optimisation des Systèmes
France

T
H
E
S
E

RESUME

Le problème global (*Simultaneous Management of Energy Production and Consumption : SMEPC*) que nous abordons dans le cadre de ce projet, concerne la gestion synchrone sur une plateforme expérimentale appelée PAVIN (Plateformes Auvergne pour Véhicules intelligents) d'une flotte de petits véhicules électriques équipés de cellules hydrogène, qui sont nécessaires pour effectuer des tâches logistiques locales dans une zone limitée, et d'une micro-usine de production d'hydrogène chargée du remplissage périodique en hydrogène de ces véhicules. On considère 1 seul véhicule, nécessaire pour effectuer des tâches selon un ordre préétabli. Le véhicule commence sa tournée avec une certaine charge de carburant hydrogène, et son réservoir a une capacité limitée. Il doit donc retourner périodiquement à la micro-usine pour se recharger en carburant (de l'hydrogène). La micro-usine a une capacité de production/stockage limitée, qui dépend de l'ensoleillement.

Notre objectif est de programmer simultanément les opérations de ravitaillement du véhicule et l'activité de production/stockage de la micro-usine en minimisant le coût de production et la durée de la tournée. Ce problème global est complexe car il implique à la fois des caractéristiques liées à l'incertitude des prévisions météorologiques, au comportement autonome du véhicule mais aussi liées à la synchronisation des recharges du véhicule et de la production d'hydrogène. On modélise d'abord notre problème par des programmes linéaires. Ensuite, pour résoudre le problème **SMEPC** plusieurs méthodes d'optimisation sont abordées à savoir : la méthode d'optimisation de type programmation dynamique qu'on nomme ici **DPS_SMEPC** et la méthode d'optimisation de type heuristique qu'on appelle ici **Pipe-line VD_PM**. Pour finir, on conçoit un estimateur de coûts à l'aide de procédés d'approximation de type réseau de neurones.

Nous modélisons **SMEPC** par un programme linéaire en nombre entiers mixtes (MIP). Les variables de décisions sont associées à la recharge du véhicule et à la production d'hydrogène. Mais, comme il implique un trop grand nombre de variables, on le décompose en deux sous-problèmes distincts. Le premier contrôle la tournée du véhicule, et le second traite de la gestion de la micro-usine. Ils sont également traités par des programmes linéaires reliés entre eux par un mécanisme de synchronisation.

SMEPC est traité par programmation dynamique. Le schéma de programmation dynamique (DPS) tend à impliquer en pratique un nombre excessivement élevé d'états. Nous introduisons des dispositifs de filtrage : basés sur l'anticipation des incohérences en définissant des règles logiques, basés sur une estimation optimiste sur la base du pré-calcul d'une solution initiale réalisable, et, basés sur l'utilisation de mécanismes de dominance. Une partie de l'étude est consacrée à une évaluation de la puissance de ces processus de filtrage.

Dans le souci de faire ressortir la dimension **collaborative** de notre problème, nous traitons

SMEPC par le biais d'une heuristique en le décomposons en deux sous-problèmes distincts, le problème **Vehicle-Driver (VD)** qui consiste à déterminer une **stratégie de recharge** du véhicule, et le problème **Production-Manager (PM)** qui consiste à déterminer une **stratégie de production** de la micro-usine. En effet, la solution du problème véhicule est considérée comme une entrée du problème production. La stratégie de production est calculée en tenant compte de la tournée trouvée par le problème véhicule.

Pour pouvoir prédire rapidement le coût de la solution optimale d'une instance, on conçoit un estimateur. Ce dernier a été construit à l'aide de réseaux de neurones qui prennent en entrées les données de l'instance et fournissent en sortie une estimation de la valeur optimale. Pour réaliser la phase d'apprentissage et de test des réseaux de neurones, on utilise un ensemble de 6000 instances.

Mots clés : Ordonnancement, Gestion de l'Energie, Programmation Dynamique, Réseaux de Neurones .

ABSTRACT

The global problem (*Simultaneous Management of Energy Production and Consumption : SMEPC*) that we are addressing in this project, concerns the synchronous management on a platform called PAVIN (Plateformes Auvergne pour Véhicules intelligents) of a fleet of small, experimental electric vehicles equipped with hydrogen cells, which are required to perform logistical tasks in a limited area, and a micro-hydrogen production plant for filling the hydrogen tank of these vehicles. We consider only 1 vehicle, necessary to perform tasks in an order of predefined. The vehicle starts its tour with a certain amount of hydrogen fuel, and its tank has limited capacity. It must therefore return periodically to the micro-plant to recharge in fuel (hydrogen). The micro-plant has a limited production/storage capacity, which depends on sunlight.

Our objective is to simultaneously schedule the vehicle refueling operations and the production/storage activity of the production/storage activity of the microfactory while minimizing the production cost and the duration of the tour. This global problem is complex because it involves both characteristics related to the uncertainty of the weather forecast, the autonomous behavior of the vehicle but also related to the synchronization of the vehicle's recharging and hydrogen production. We first model our problem by linear programs. Then, to solve the problem **SMEPC** several methods of optimization are approached namely : the method of optimization of type dynamic programming that we call here **DPS_SMEPC** and the optimization method of heuristic type that we call here **Pipe-line VD_PM**. Finally, we design a cost estimator using neural network approximation processes.

We model **SMEPC** by a mixed integer linear program (MIP). The decision variables are associated with vehicle charging and hydrogen production. However, since it involves too many variables, it is decomposed into two distinct subproblems. The first one controls the vehicle tour, and the second one deals with the management of the microfactory. They are also handled by linear programs linked together by a synchronization mechanism.

SMEPC is processed by dynamic programming. The Dynamic Programming Scheme (DPS) tends to involve in practice an excessively high number of states. We are introducing filtering devices : based on the anticipation of inconsistencies by defining logical rules, based on an optimistic estimation based on the pre-calculation of a feasible initial solution, and, based on the use of dominance mechanisms. Part of the study is devoted to an evaluation of the power of these filtering processes.

In order to bring out the **collaborative** dimension of our problem, we treat **SMEPC** through a heuristic by decomposing it into two sub-problems distinct, the problem **Vehicle-Driver (VD)** which consists in determining a **recharging strategy** of the vehicle, and the problem **Production-Manager**

(PM) which consists in determining a **production strategy** for the micro-plant. Indeed, the solution of the vehicle problem is considered as an input to the production problem. The production strategy is calculated taking into account the route found by the vehicle problem.

In order to quickly predict the cost of the optimal solution of an instance, an estimator is designed. This estimator is built using neural networks that take as input the data of the instance and provide as output an estimate of the optimal value. To carry out the learning and testing phase of the neural networks, a set of 6000 instances is used.

Keywords : Scheduling, Energy Management, Dynamic Programming, Neural Networks .

FINANCEMENTS

Ce travail de recherche est cofinancé par le Labex IMobS3 et le FEDER-Région Auvergne.



PUBLICATIONS

Les travaux de recherches réalisés durant cette thèse ont donné lieu aux publications présentées ci-dessous.

Journaux

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. Synchronizing Energy Production and Vehicle Routing. *RAIRO - Operations Research*, 2021.

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. Pipe-Lining Dynamic Programming Processes in Order to Synchronize Management Energy Production and Consumption. *RAIRO - Operations Research*, 2021.

Conférences internationales

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. Simultaneous Management of Energy Production and Consumption. *Control, Decision and Information Technologies (CoDIT)*, Prague, Republique Czech, 2020.

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. Pipe-Lining Dynamic Programming Processes in Order to Synchronize Management Energy Production and Consumption. *FedCSIS*, 2020.

Conférences nationales

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. MIP for the simultaneous management of energy production and consumption. *Société française de recherche opérationnelle et d'aide à la décision (ROADEF)*, Monpellier, France 2020.

Invitations aux conférences

Fatiha Bendali, Jean Mailfert, Eloise Yollande Mole Kamga, Alain Quilliot et Hélène Toussaint. Simultaneously dealing with renewable (Hydrogen) energy production and consumption. *NICST*, Bordeaux, France 2019.

Collaboration internationale

Dans le cadre du projet GEO-SAFE, j'ai effectué un séjour de recherche de Mai 2019 à Juillet 2019 à Melbourne en Australie. Ce travail a été réalisé dans le cadre du projet européen H2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange European project 691161 "GEO-SAFE" Geospatial based Environment for Optimisation Systems Addressing Fire Emergencies, 691161, 2020.

TABLE DES MATIÈRES

Liste des algorithmes	10
Table des figures	13
Liste des tableaux	15
1 Introduction générale	16
1.1 Contexte et motivations	17
1.2 Objectifs de la thèse	25
1.3 Plan de la thèse	26
2 État de l'art du problème SMEPC	28
2.1 Introduction	29
2.2 Recherche Opérationnelle et Énergie	29
2.3 Problématique générale de synchronisation	40
2.4 Conclusion	42
3 Etat de l'art sur les méthodes	45
3.1 Introduction	46
3.2 Généralités sur la modélisation	46
3.3 Programmation dynamique	47
3.4 Apprentissage automatique	63
3.5 Conclusion	76
4 Programmation linéaire en nombres entiers mixtes de SMEPC	77
4.1 Introduction	79
4.2 Le problème SMEPC : <i>Synchronous Management of Energy Production and Consumption</i>	79
4.3 Formulation mathématique	85
4.4 Formulation par programmation linéaire en variables mixtes : $MILP_{SMEPC}$	88
4.5 Relaxation linéaire de la formulation $MILP_{SMEPC}$ et contraintes additionnelles STC et EC : $RMILP_{SMEPC}$	89
4.6 Complexité du modèle SMEPC	93
4.7 Expérimentations numériques	93

4.8 Conclusion	112
4.9 Annexes	112
5 Schéma de programmation dynamique pour résoudre SMEPC	115
5.1 Introduction	117
5.2 Un algorithme de programmation dynamique pour SMEPC : DPS_SMEPC	117
5.3 Un schéma d'approximation polynomial (PTAS) : filtrage par arrondi (Rounding)	129
5.4 Mécanismes de filtrage	131
5.5 Description d'une heuristique rapide <i>He</i> pour résoudre SMEPC	139
5.6 Expérimentations numériques	142
5.7 Conclusion	155
5.8 Annexes	155
6 Schéma collaboratif pour résoudre SMEPC	158
6.1 Motivations	160
6.2 Le problème du véhicule : <i>Vehicle-Driver</i> (VD)	160
6.3 Le problème de production : <i>Production-Manager</i> (PM)	163
6.4 L'algorithme de programmation dynamique DPS_VD	165
6.5 L'algorithme de programmation dynamique DPS_PM	172
6.6 Collaboration Demandeur/Producteur pour SMEPC : le schéma Pipe-line VD_PM	181
6.7 Expérimentations numériques	185
6.8 Conclusion	198
6.9 Annexes	198
7 Estimation des coûts de production par apprentissage	204
7.1 Introduction	206
7.2 Réseaux de neurones dont les entrées sont des données brutes	209
7.3 Réseaux de neurones dont les entrées sont des indicateurs	216
7.4 Expérimentations numériques	225
7.5 Conclusion	242
8 Conclusion générale	244
8.1 Récapitulatif de notre contribution	245
8.2 Perspectives	246

LISTE DES ALGORITHMES

1	Backward_DPS	53
2	DPS_SMEPC	128
3	DPS_SMEPC(K)	130
4	Estimation-énergie(j)	133
5	Cout minimal production	136
6	Greedy_SMEPC	138
7	Search_BSUP	147
8	Veh_Bellman_Update	168
9	DPS_VD	169
10	Stratégie-de-recharge-primaire	170
11	Stratégie-de-recharge-réduite	171
12	Prod_Bellman_Update	176
13	DPS_PM	177
14	Greedy_PM	180
15	Filtered_DPS_PM	182
16	Pipe-line VD_PM	184
17	T_Reconstruction	184

TABLE DES FIGURES

1.1	Plateforme PAVIN « Véhicule Urbains »	19
1.2	Plateforme PAVIN VMN « Véhicule en Milieu Naturel »	19
1.3	Plateforme PAVIN BP « Brouillard et Pluie »	19
1.4	Une voiture hybride	22
1.5	Première station à hydrogène de Clermont-Ferrand	24
2.1	Problèmes incluant tournée de véhicules et gestion de leurs alimentations	30
3.1	Arcs associés aux Transitions	51
3.2	Structure Générale du Réseau d'État "Production"	51
3.3	Les nœuds du Réseau des Etats du problème du sac à dos.	56
3.4	Regroupement des états.	57
3.5	Un arc-transition non déterministe.	59
3.6	Graphe des Etats Cycliques.	62
3.7	Une couche de 5 neurones complétement reliée à une couche de 3 neurones.	65
3.8	Une couche de 5 neurones partiellement reliée à une couche de 3 neurones.	65
3.9	Représentation des poids et du biais.	66
3.10	Calcul des valeurs d'entrée et de sortie d'un neurone.	66
3.11	Poids et biais entre deux couches de neurones non unitaires.	67
3.12	Une matrice des poids W et un vecteur des biais B	67
3.13	Affichage des poids.	69
3.14	Représentation graphique des poids associés à la Figure (3.13).	69
3.15	Représentation d'un réseau avec 2 couches partiellement reliées.	72
3.16	Représentation du réseau de la figure (3.15) en utilisant exclusivement des layers tf.keras Dense et Concatenate.	73
3.17	Le réseau de la figure (3.8) représenté avec des layers Dense et Concatenate.	74
3.18	Exemple de réseau dans lequel des poids fixes et dépendant de l'instance interviennent.	75
3.19	Figure (3.18) reformulé avec des layers Dense et Multiply.	76
4.1	Le dépôt et la micro-usine	79
4.2	Notations concernant les valeurs de temps et d'énergie	80
4.3	Une tournée de véhicule	80

4.4	Intervalle de temps correspondant à une période	81
4.5	Une stratégie de production	81
4.6	Coûts de production	82
4.7	Une solution possible d'une instance de SMEPC	83
4.8	Décomposition du problème SMEPC	84
4.9	Plan contenant les stations	94
4.10	Représentation graphique du CPU et du gap des tableaux (4.9) et (4.10)	107
4.11	Représentation graphique du CPU et du gap des tableaux (4.11) et (4.11)	107
4.12	Représentation graphique du CPU et du gap des tableaux (4.13) et (4.14)	111
4.13	Représentation graphique du CPU et du gap des tableaux (4.15) et (4.16)	111
4.14	Une tournée du véhicule d'une instance de SMEPC	114
5.1	Positionnements relatifs du véhicule par rapport à une station	118
5.2	Synchronisation du problème VD et du problème PM	126
5.3	Stratégie de génération des états du DPS_SMEPC	127
5.4	Impact des mécanismes de filtrage sur le DPS_SMEPC	131
5.5	Une instance	141
5.6	Le graphe G	141
5.7	Les données de la partie production de l'instance A	143
5.8	Les données de la partie recharge de l'instance A	143
5.9	La solution de l'instance A	144
5.10	Les données de la partie production de l'instance B	145
5.11	Les données de la partie recharge de l'instance B	145
5.12	La solution de l'instance B	146
5.13	Représentation graphique du CPU et du gap du tableau (5.3)	149
5.14	Représentation graphique du CPU et du gap du tableau (5.4)	150
5.15	Représentation graphique du CPU et du gap du tableau (5.5)	153
5.16	Représentation graphique du CPU et du gap du tableau (5.6)	154
6.1	Schéma du Pipe-line VD_PM	183
6.2	La solution de l'instance A	186
6.3	La solution de l'instance B	187
6.4	Représentation graphique du CPU et du gap du tableau (6.12)	190
6.5	Représentation graphique du CPU et du gap du tableau (6.14)	192
6.6	Représentation graphique du CPU et du gap du tableau (6.13)	193
6.7	Le sous-graphe Useful VD_Subgraph	203
7.1	Réseau SIMPLE_TYPE	211
7.2	Réseau SIMPLE_PERIODE	211
7.3	Réseau MIXTE	214
7.4	Réseau INDIC_TEMPS prédisant le numéro de période de la dernière recharge	221
7.5	Réseau de neurones prédisant le coût $Cost_A$	224
7.6	Réseau de neurones prédisant le coût $Cost_P$	224
7.7	Réseau INDIC_COUT prédisant le coût $COST$	225
7.8	Coût moyen de production par ordre croissant et regroupé par paquet de 100 instances	227

7.9 Coût fixe moyen et coût variable moyen de production regroupé par paquet de 100 instances	228
7.10 Moyenne du nombre de stations regroupé par paquet de 100 instances	228
7.11 Moyenne du nombre de recharges regroupé par paquet de 100 instances	229
7.12 Évolution du temps maximal moyen regroupé par paquet de 100 instances	230
7.13 Capacité moyenne du réservoir et de la citerne regroupé par paquet de 100 instances	230
7.14 Le gap du réseau SIMPLE_TYPE	234
7.15 Résultats du réseau de neurones SIMPLE_TYPE	234
7.16 Le gap du réseau SIMPLE_PERIODE	235
7.17 Résultats du réseau SIMPLE_PERIODE	236
7.18 Le gap du réseau MIXTE_COUT	237
7.19 Résultats du réseau MIXTE_COUT	237
7.20 Le gap du réseau INDIC_COUT	238
7.21 Résultats du réseau de neurones INDIC_COUT	239
7.22 Le gap du réseau MIXTE_TEMPS	240
7.23 Résultats du réseau MIXTE_TEMPS	240
7.24 Le gap du réseau INDIC_TEMPS	241
7.25 Résultats du réseau de neurones INDIC_TEMPS	242

LISTE DES TABLEAUX

1.1	Comparaison entre les procédés de fabrication de l'hydrogène	20
2.1	Récapitulatif de l'état de l'art du problème SMEPC	43
2.2	Récapitulatif de l'état de l'art du problème SMEPC (suite)	44
4.1	Entrées du problème SMEPC	84
4.2	Linéarisation des contraintes	89
4.3	Caractéristiques des instances du paquet INST_VAR	97
4.4	Caractéristiques des instances du paquet INST_CTE	98
4.5	Caractéristiques des solutions des instances du paquet INST_VAR	99
4.6	Caractéristiques des solutions des instances du paquet INST_CTE	100
4.7	Résultats de $MILP_{SMEPC}$ sur INST_VAR. Le temps limite est 3 heures sur 8 threads	101
4.8	Résultats de $MILP_{SMEPC}$ sur INST_CTE. Le temps limite est 3 heures sur 8 threads	102
4.9	Impact des contraintes STC sur les résultats de $RMILP_{SMEPC}$ sur INST_VAR	104
4.10	Impact des contraintes EC sur les résultats de $RMILP_{SMEPC}$ sur INST_VAR	105
4.11	Impact des contraintes STC sur les résultats de $RMILP_{SMEPC}$ sur INST_CTE	106
4.12	Impact des contraintes EC sur les résultats de $RMILP_{SMEPC}$ sur INST_CTE	106
4.13	Résultats de $MILP_{SMEPC}$ enrichi de STC sur INST_VAR. Le temps limite est 1 heure en mode <i>single-threads</i>	108
4.14	Résultats de $MILP_{SMEPC}$ enrichi de EC sur INST_VAR. Le temps limite est 1 heure en mode <i>single-threads</i>	109
4.15	Résultats de $MILP_{SMEPC}$ enrichi de STC sur INST_CTE. Le temps limite est 1 heure en mode <i>single-threads</i>	110
4.16	Résultats de $MILP_{SMEPC}$ enrichi de EC sur INST_CTE. Le temps limite est 1 heure en mode <i>single-threads</i>	110
5.1	Liste des pré-conditions du DPS_SMEPC	125
5.2	Simulation de l'évolution du temps DPS	126
5.3	Les instances du paquet INST_VAR : recherche d'une borne supérieure de qualité pour le DPS_SMEPC.	148
5.4	Les instances du paquet INST_CTE : recherche d'une borne supérieure de qualité pour le DPS_SMEPC.	149

5.5 Les instances du paquet INST_VAR : impact des mécanismes de filtrage du DPS_SMEPC.	153
5.6 Les instances du paquet INST_CTE : impact des mécanismes de filtrage du DPS_SMEPC.	154
6.1 Entrées du problème VD	161
6.2 Une instance du modèle VD	162
6.3 La stratégie de recharge optimale de l'instance du modèle VD du tableau (6.2)	162
6.4 Une instance du modèle VD	163
6.5 La stratégie de recharge optimale de l'instance du modèle VD du tableau (6.4)	163
6.6 Entrées du problème PM	164
6.7 La stratégie de recharge primaire de l'instance du modèle VD du tableau (6.2)	170
6.8 La stratégie de recharge primaire de l'instance du modèle VD du tableau (6.4)	170
6.9 La stratégie de recharge réduite de l'instance du modèle VD du tableau (6.2)	171
6.10 La stratégie de recharge réduite de l'instance du modèle VD du tableau (6.4)	171
6.11 Une stratégie de recharge réduite	174
6.12 Les instances du paquet INST_VAR : Comparaison des gaps et du temps CPU du Pipe- Line VD_PM.	190
6.13 Les instances du paquet INST_VAR : impact des mécanismes de filtrage.	191
6.14 Les instances du paquet INST_CTE : comparaison des gaps et du temps CPU du Pipe- Line VD_PM.	192
6.15 Les instances du paquet INST_CTE : impact des mécanismes de filtrage.	193
6.16 Les instances du paquet INST_VAR : Comparaison des deux bornes supérieures utilisées pour le filtrage du DPS_SMEPC.	195
6.17 Les instances du paquet INST_CTE : Comparaison des deux bornes supérieures utilisées pour le filtrage du DPS_SMEPC.	196
6.18 Les instances du paquet INST_VAR : impact des mécanismes de filtrage du DPS_SMEPC.	197
6.19 Les instances du paquet INST_CTE : impact des mécanismes de filtrage du DPS_SMEPC.	198
6.20 Entrées du <i>Useful VD_Subgraph</i>	203
7.1 Glossaire	206
7.2 Entrées utilisées pour construire nos réseaux de neurones	207
7.3 Apprentissage du coût de production : Moyenne de la valeur absolue des gaps des données de test et d'apprentissage	231
7.4 Apprentissage du numéro de période de la dernière recharge : Moyenne de la valeur absolue des gaps des données de test et d'apprentissage	231
7.5 Comparaison des fonctions d'activation du réseau SIMPLE_PERIODE	233
8.1 Synthèse de quelques variants du problème SMEPC	247

CHAPITRE 1

INTRODUCTION GÉNÉRALE

Sommaire

1.1 Contexte et motivations	17
1.1.1 Contexte de travail	17
1.1.2 Hydrogène	18
1.1.3 Véhicules à hydrogène	21
1.1.4 Liste de projets	23
1.2 Objectifs de la thèse	25
1.3 Plan de la thèse	26

1.1 Contexte et motivations

1.1.1 Contexte de travail

Ce travail de thèse a été réalisé au sein du LIMOS (Laboratoire d’Informatique, de Modélisation et d’Optimisation des Systèmes). Le LIMOS est une Unité Mixte de Recherche (UMR 6158) de l’Université Clermont Auvergne situé à Clermont-Ferrand en France. Les résultats qu’on présentera dans ce rapport de recherche sont le fruit d’une collaboration entre l’axe MAAD et l’axe ODPS qui font partie des trois principaux axes de recherche du LIMOS. Ces axes sont respectivement l’axe Modèles et Algorithmes de l’Aide à la Décision (MAAD), l’axe Outils Décisionnels pour la Production et les Services (ODPS) et l’axe Systèmes d’Information et de Communication (SIC). Le positionnement scientifique de chacun de ces axes est le suivant :

- Les chercheurs de MAAD s’intéressent à la résolution mathématique et algorithmique de problèmes d’optimisation [95] en Recherche Opérationnelle. Ils explorent plusieurs approches en particulier les algorithmes d’approximation (algorithmes qui produisent en temps polynomial des solutions dont on peut garantir la qualité par rapport à la solution optimale) et les méthodes polyédrales (approches qui consistent à ramener chaque problème à la résolution d’un programme linéaire).
- Les chercheurs d’ODPS s’occupent de la modélisation de systèmes complexes et de l’implémentation des méthodes aidant à la prise de décisions. La plupart du temps, ils identifient dans un premier temps des problématiques nouvelles et par la suite conçoivent des méthodes d’optimisation capables de les traiter efficacement.
- Les chercheurs de SIC s’intéressent à la collecte (à travers par exemple des mécanismes de communication sans fil), la gestion et l’analyse de grande masse de données en utilisant des techniques de fouille de données et d’apprentissage automatique.

Le LIMOS est membre du Laboratoire d’Excellence pour la Mobilité Innovante des personnes, des biens et des machines dont l’acronyme est Labex IMobS3. Le Labex IMobS3 a cofinancé cette thèse à hauteur d’environ 1/3 du financement total et a utilisé un budget adossé sur des financements provenant d’une aide de l’état et gérés par l’Agence Nationale de la Recherche (ANR) au titre du programme Investissements d’Avenir. Le Laboratoire d’Excellence IMobS3 conçoit des systèmes respectueux de l’environnement pour une mobilité innovante des personnes (par exemple l’équipement de fauteuils pour personnes à mobilité réduite, et l’équipement de cannes blanches pour les personnes malvoyantes), des biens (par exemple le routage de véhicules autonomes pour le transport de marchandise) et des machines (par exemple le routage de drones). De plus, ils étudient les interactions entre des systèmes de mobilité innovante et leur environnement, ainsi que l’impact économique lié à la mise en place de tels systèmes, et l’impact sur l’Homme ou sur la Société comme par exemple les niveaux de prix d’accès au système proposés à l’utilisateur. Le but est de satisfaire les besoins des utilisateurs et d’augmenter la capacité de service en automatisant le plus possible certains travaux. Un autre axe d’étude du Labex IMobS3 est la recherche de procédés de production d’énergie renouvelable pour alimenter les systèmes de mobilité innovante. Une énergie est dite renouvelable lorsqu’elle provient de sources que la nature renouvelle en permanence. Les systèmes de mobilité innovante conçus sont testés au sein de plateformes expérimentales appelées PAVIN.

On a travaillé dans le cadre de ce projet, avec des chercheurs de PAVIN qui nous ont renseigné sur les mécanismes de fabrication des énergies renouvelables. Il existe trois plateformes PAVIN : la plateforme **PAVIN VU « Véhicule Urbains »** (Voir figure 1.1), la plateforme **PAVIN VMN « Véhicule en Milieu Naturel »** (Voir figure 1.2) et la plateforme **PAVIN BP « Brouillard et Pluie »** (Voir figure 1.3). Le rôle de chaque plateforme est le suivant :

- La plateforme **PAVIN VU** simule une mini-ville disposant de différents véhicules électriques et travaille sur les procédés permettant d'accroître le degré d'autonomie de ces derniers. Ce site expérimental original situé à Clermont-Ferrand (France), a été conçu pour le déplacement de véhicules autonomes.
- La plateforme **PAVIN VMN** se trouve sur le site de Montoldre de l'Institut National de Recherche en Sciences et Technologies pour l'Environnement et l'Agriculture (IRSTEA) dans l'Allier. Ce site de 70 hectares permet de faire évoluer divers véhicules à haute vitesse.
- La plateforme **PAVIN BP** du Centre d'Etudes et d'Expertise sur les Risques, l'Environnement, la Mobilité et l'Aménagement (CEREMA) est composée d'une part d'un tunnel permettant de simuler des conditions atmosphériques dégradées tant pour le brouillard que la pluie (plateforme unique au plan national) et d'autre part d'un dispositif mobile pouvant être déployé sur **PAVIN VU** et **PAVIN VMN**. Cette plateforme se situe à Clermont-Ferrand.

Toutes ces plateformes se situent dans la région AURA, Auvergne-Rhône-Alpes.

Finalement, le FEDER-Region Auvergne a cofinancé cette thèse à hauteur d'environ 2/3 du financement total et a utilisé une aide de l'Union Européenne via les Fonds Européen de Développement Régional (FEDER-Région Auvergne).

En Auvergne, on distingue trois types de filières de production d'énergie :

- La filière classique qui regroupe les centrales nucléaires et thermiques.
- La filière d'énergie renouvelable thermique (bois énergie, pompes à chaleur, solaire, valorisation thermique des déchets et du biogaz, etc).
- La filière d'énergie renouvelable électrique (hydraulique, éolien, photovoltaïque, valorisation électrique des déchets et du biogaz, etc).

Notre projet de thèse fait partie d'un vaste projet dont l'objectif est la vulgarisation de l'utilisation des énergies renouvelables, plus précisément de l'**hydrogène** solaire.

1.1.2 Hydrogène

L'intérêt pour l'hydrogène vient du fait qu'à ce jour, les principales sources d'énergie sont les ressources fossiles par exemple le pétrole, le gaz naturel et le charbon. Ces ressources fossiles mettent des millions d'années (environ 250 millions) à se former, elles sont très polluantes, elles sont présentes en quantités limitées et se renouvellent beaucoup plus lentement que nous les utilisons. En France, la filière hydrogène n'est pas encore très développée, mais la société Air liquide et la société MYRTE basée en Corse fabriquent déjà l'hydrogène.



FIGURE 1.1 – Plateforme PAVIN « Véhicule Urbains » [48].



FIGURE 1.2 – Plateforme PAVIN VMN « Véhicule en Milieu Naturel » [48].

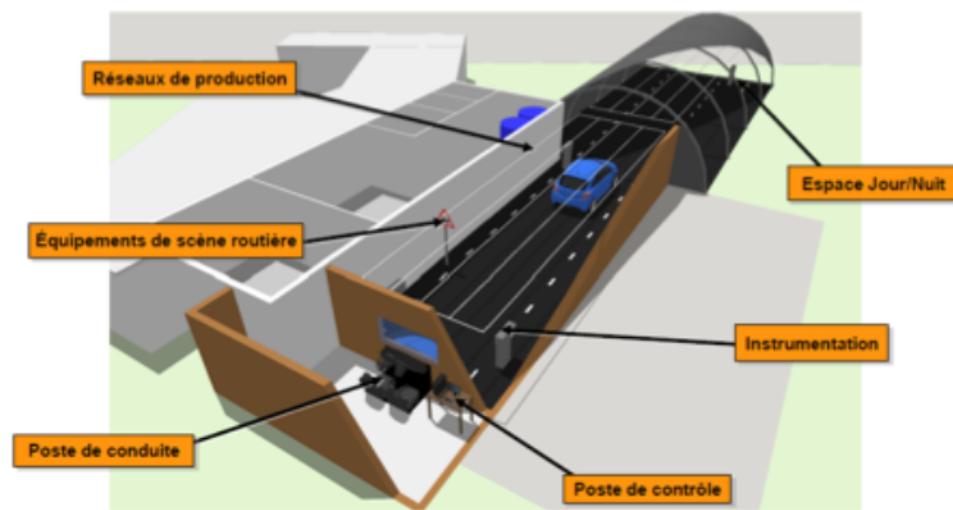


FIGURE 1.3 – Plateforme PAVIN BP « Brouillard et Pluie » [48].

Actuellement, il existe trois procédés de fabrication de l'hydrogène :

- L'hydrogène peut être produite par électrolyse de l'eau (H_2O). L'électrolyse de l'eau (H_2O) consiste à casser la molécule d'eau à l'aide de l'électricité afin d'obtenir de l'hydrogène pur

(technique utilisée dans les boosters des fusées Ariane). Pour le faire, on a besoin d'une photocatalyse et d'un catalyseur. On utilise l'électrolyse alcaline ou l'électrolyse basique dont la différence se trouve au niveau de l'acidité de l'eau. La quantité d'hydrogène produite est $100 \text{ m}^3/\text{heure}$. Cette technique est utilisée par MYRTE.

- L'hydrogène peut être produite par vaporeformage du méthane (CH_4). Le vaporeformage du méthane (CH_4) consiste à produire de l'hydrogène en utilisant le méthane, une source de chaleur et de l'eau, mais l'énergie produite n'est pas renouvelable. Par contre la quantité d'hydrogène fabriquée est énorme ($100.000 \text{ m}^3/\text{heure}$). Cette technique est utilisée par la société Air liquide. L'inconvénient de cette méthode est qu'elle produit beaucoup de CO_2 .
- L'hydrogène peut être produite par photolyse de l'eau. Lors de la photolyse, l'énergie de la lumière solaire provoque la cassure des molécules d'eau. On utilise des cellules photo-électrochimiques (PEC), constituées d'électrodes photosensibles et plongées dans l'eau. La lumière absorbée génère des électrons (négatifs) et des charges positives formées par des manques d'électrons appelés trous. En présence de ces paires électrons-trous, les molécules d'eau subissent une réaction d'oxydo-réduction. Hydrogène et oxygène partent chacun de leur côté sous forme gazeuse, H_2 et O_2 . Cette technique donne de meilleure quantité d'énergie (beaucoup de H_2) que l'électrolyse de l'eau mais il faudrait une grande surface pour pouvoir capter du soleil. Cette technique est à un stade expérimental et a été le projet de thèse en Génie des procédés de **Victor Gattepaille** sous la direction de **Jean Francois Cornet**, en partenariat avec Institut Pascal de Novembre 2017 à Janvier 2021.

Procédés Caract.	Photolyse	Vaporeformage CH_4	Electrolyse de l'eau
Réalisé de nos jours	stade expérimental	oui	oui
Nombre de m^3 par heure produite	Supérieur à 100	100.000	100
Difficultés	Nécessite trop de soleil	Pas renouvelable	Nécessite trop d'électricité
Ingrédients	Eau, soleil	Méthane, eau, chaleur	Eau, électricité
Taux de pollution	Faible	Forte	Celui pour produire l'électricité
Société	En laboratoire	Air liquide	MYRTE

TABLE 1.1 – Comparaison entre les procédés de fabrication de l'hydrogène.

Le tableau (1.1) est un récapitulatif de l'étude comparative qu'on a faite entre les caractéristiques (appelées Caract.) des procédés de fabrication de l'hydrogène (appelés Procédés).

Une énergie « propre » est une énergie qui produit une faible quantité de CO_2 lors de son utilisation et de sa fabrication. Concernant son utilisation, l'hydrogène est une énergie « propre » mais la façon dont on produit cet hydrogène n'est pas toujours « propre ». L'un des avantages de l'hydrogène est sa bonne autonomie car par exemple les modèles de voiture à hydrogène aujourd'hui peuvent parcourir plus de 500 kilomètres. L'inconvénient de l'hydrogène est sa faible masse volumique. Pour conserver une grande quantité d'hydrogène on a ainsi besoin de très grands réservoirs. A cet effet, des réservoirs à 350 bar ont été conçus car en augmentant la pression dans un réservoir on diminue le volume de celui-ci accroissant la masse qu'il peut contenir.

Il existe de nombreux cas d'applications de l'hydrogène, on a :

- Des applications mobiles : l'hydrogène peut alimenter des véhicules équipés de moteurs à combustion fonctionnant au gaz. Par ailleurs, un réservoir d'hydrogène peut-être associé à une pile combustible pour améliorer l'autonomie de véhicules électriques. De plus, la mise en place de flottes captives (par exemple celles de La Poste) fonctionnant à l'hydrogène permet d'augmenter la productivité tout en diminuant les émissions de CO_2 sur le lieu d'utilisation. Les principales applications concernent les flottes de chariots élévateurs d'entrepôts de logistique ou encore les flottes de véhicules de transport de bagages dans les aéroports. L'hydrogène peut aussi être utilisé comme carburant pour les fusées comme c'est le cas pour la Fusée Européenne Ariane 5.
- Des applications stationnaires qui consistent à stocker de l'énergie dans des bâtiments en assurant une fourniture d'électricité et de chaleur grâce à la cogénération en produisant simultanément deux types d'énergie, et cela dans une même centrale. L'hydrogène permet une grande autonomie et est aussi particulièrement adapté à la fourniture d'énergie pour des équipements stationnaires éloignés du réseau (ou en attente de raccordement). C'est notamment le cas pour les antennes relais de téléphonie mobile.
- Des applications industrielles : l'hydrogène est un composant chimique très employé dans l'industrie. De nombreuses applications industrielles l'utilisent pour produire différents matériaux. En électronique, l'hydrogène sert à la fabrication de composants. Il assure une excellente protection contre les impuretés. En chimie industrielle, l'hydrogène peut être associé à de l'azote pour fabriquer de l'ammoniac, une base des engrains. C'est un réactif qui entre dans la composition des fibres textiles comme le nylon et de diverses matières plastiques. Dans l'industrie du verre, l'hydrogène est indispensable à la fabrication du verre plat utilisé notamment pour les écrans plats. L'hydrogène est employé en métallurgie pour produire des pièces mécaniques.

Dans le cadre de cette thèse, on s'intéresse aux applications mobiles, plus particulièrement aux véhicules. L'hydrogène sera chargé dans des piles à combustible contenues dans des **véhicules à hydrogène**.

1.1.3 Véhicules à hydrogène

Habituellement, lorsqu'on parle de véhicules fonctionnant avec de l'énergie renouvelable, on pense tout de suite aux véhicules électriques. Or, les véhicules électriques utilisent de nos jours des batteries électriques qui sont composées de lithium-ion. Ces batteries trop chères, constituent une catastrophe écologique car elles sont non recyclables. De plus, leur autonomie est faible et leur recharge dure trop longtemps. Ce sont là quelques raisons pour lesquelles de nombreuses institutions se tournent vers l'Hydrogène. Actuellement, la communauté scientifique s'intéresse à l'hydrogène comme source d'énergie « propre ». Le 5 septembre 2021, les chercheurs du Plasma Science and Fusion Center (PSFC) du MIT ont réussi un test d'une nouvelle génération de réacteur qui fait un nouveau genre de fusion nucléaire en utilisant comme combustible de l'hydrogène. Actuellement, on a un verrou technologique pour la fabrication d'hydrogène « propre » car il faut trouver le bon catalyseur pour synthétiser la molécule d'hydrogène via un procédé « propre ». Le vecteur énergétique de l'hydrogène est la pile à combustible qui est plus efficace et plus « propre » qu'une batterie électrique.

Les premiers véhicules électriques à hydrogène de série ont été commercialisés à partir de 2013 (Hyundai ix35, Toyota Mirai). Un exemple de véhicule utilisant l'hydrogène est présenté à la figure (1.4).



FIGURE 1.4 – Une voiture hybride : véhicule Kangoo ZE H2 premier utilitaire à l'hydrogène réservoir d'hydrogène de 76 litres (compressé à 350 bars, ce qui nous donne un poids d'environ 1,8kg qu'on recharge en 6 minutes), 27 000 euros : le coût de la pile mais sans la voiture .

Par rapport à des modèles 100 % électriques, ces voitures sont adaptées pour des parcours longue distance. Moins de cinq minutes sont nécessaires pour les recharger et offrir une autonomie de 500 km. A l'usage, elles ne produisent aucune pollution : aucun gaz à effet de serre et aucune particule.

Nous nous intéressons aux véhicules à hydrogène, et, pour mieux les cerner, on s'attarde sur leurs moteurs, leur autonomie, leurs niveaux de pollution, l'étendue de leurs gammes et leurs prix.

Plusieurs types de véhicules peuvent prétendre à l'appellation « véhicule à hydrogène ». On distingue tout d'abord les voitures embarquant une pile à combustible qui utilise de l'énergie cinétique pour alimenter un moteur électrique. Il existe également des modèles avec un moteur à hydrogène qui est alimenté grâce à des réservoirs sous pression stockant de l'hydrogène et fonctionnant sur le même principe qu'un moteur à explosion. L'autonomie d'un véhicule à hydrogène dépend principalement de la capacité du réservoir à hydrogène et du rendement de la pile à combustible. Les modèles les plus performants affichent un rendement de l'ordre de 80 %, permettant d'offrir une autonomie théorique supérieure à 500 kilomètres. À l'heure actuelle, le record est d'ailleurs détenu par la Hyundai Nexo, qui a parcouru 778 kilomètres avec un seul plein. L'hydrogène sert à alimenter un moteur électrique. La voiture à hydrogène ne pollue pas à l'usage car elle ne rejette que de la vapeur d'eau.

En 2020, La gamme de véhicules à hydrogène se limite principalement au Hyundai Nexo, à la Toyota Mirai, au Honda Clarity Fuel Cell et à la Mercedes GLC. Non seulement l'offre est limitée, mais les prix restent relativement élevés. A titre d'exemple, la Toyota Mirai coûte au minimum 78 900€, 60 000€ pour la Honda Clarity Fuel Cell et 53 100€ pour la Mercedes-Benz GLC.

En France, dans le secteur des véhicules à hydrogène, la société STEP (« Société du Taxi Électrique Parisien ») a lancé en décembre 2015 à Paris la première flotte de taxis composée uniquement de véhicules à hydrogène, sous la marque « hype », en partenariat avec Air Liquide. La société française Symbio FCell est spécialisée dans la conception et la fabrication de systèmes piles à combustible (Il existe une association pour l'hydrogène et les piles à combustible appelée AFHYPAC). Symbio FCell conçoit des kits de pile à hydrogène, pour les véhicules en particulier. Actuellement, Symbio FCell fait du recharge à moyenne pression (350 bar). La meilleure autonomie actuelle sur le marché est 100 km/10 litres d'hydrogène à 700 bars. A 700 bars l'hydrogène représente trois fois plus de volume que l'essence mais a un rendement deux fois meilleur.

Dans la section suivante, on va lister quelques projets qui participent à la vulgarisation de l'hydrogène.

1.1.4 Liste de projets

Parmi les projets qui mettent en exergue l'hydrogène on peut citer : Hympulsion, ZEV, Blue Hydrogen et PARKOSOL.

Le projet Hympulsion :

Hympulsion a été créé en novembre 2018 par Engie et Michelin pour financer et exploiter vingt **stations de distribution d'hydrogène** en Auvergne-Rhône-Alpes d'ici à 2023. En 2019 la première station à hydrogène a été installée à Clermont-Ferrand (Voir figure 1.5). La région Auvergne-Rhône-Alpes (AURA) participe aussi au développement de l'**hydrogène** comme vecteur de mobilité décarbonnée.

Le projet ZEV :

En contrôlant un tiers du capital d'Hympulsion, la région AURA accélère la réalisation du projet Zero Emission Valley (ZEV) qui vise à déployer **1 000 véhicules à pile à combustible et 15 électrolyseurs pour produire de l'hydrogène**. Ce programme de 50 millions d'euros soutenu par la Banque des territoires et l'Ademe est subventionné à hauteur de 10,1 millions d'euros par l'Union européenne. L'achat de véhicules à pile à combustible sera subventionné proportionnellement au nombre de kilomètres parcourus. ZEV veut propulser la filière hydrogène régionale, qui comprend une centaine d'acteurs, laboratoires, collectivités et entreprises comme McPhy (producteur d'électrolyseur/stockage d'énergie), Symbio FCell (producteur de pile à combustible), Ergosup (producteur d'hydrogène par électrolyse) et Ad-Venta (producteur de systèmes de stockage hydrogène embarqués).

Le projet Blue Hydrogen :

Avec sa démarche Blue Hydrogen, Air Liquide s'oriente vers une dé-carbonisation progressive de sa production d'hydrogène dédié aux applications énergétiques. Concrètement, Air Liquide a pour objectif de produire au moins 50 % de l'hydrogène nécessaire à ces applications sans rejet de CO_2 en combinant : le reformage de biogaz, l'utilisation des énergies renouvelables via l'électrolyse de l'eau et l'usage des technologies de captage et de valorisation du CO_2 émis lors de la production d'hydrogène à partir de gaz naturel essentiellement composé de méthane (CH_4). À distance parcourue égale, les voitures électriques à hydrogène permettent de diminuer les émissions de gaz à effet de serre par rapport aux véhicules à combustion.



FIGURE 1.5 – Première station à hydrogène de Clermont-Ferrand [Rue du Quebec, Zone des Ga-vranches].

Le projet PARKOSOL :

Engagée dans le développement durable et dans la production locale d'énergie renouvelable, GEG ENeR lance PARKOSOL, un projet portant sur la construction puis l'exploitation de centrales photovoltaïques en ombrières sur des parkings relais de la métropole grenobloise. L'énergie solaire est une énergie 100% renouvelable que GEG exploite partout en France et notamment en région Auvergne Rhône-Alpes. GEG ENeR a de nombreux projets de déploiement, de réalisation et d'exploitation d'énergies renouvelables telles que l'hydroélectricité, l'éolien, le biogaz et le photovoltaïque. Avec PARKOSOL, GEG ENeR veut diminuer les consommations énergétiques, les émissions de gaz à effet de serre et les polluants atmosphériques. Le projet PARKOSOL s'inscrit également dans les démarches de production locale d'énergie renouvelable visées par le plan Air Energie Climat de la collectivité Grenoble Alpes Métropole.

Notre projet :

Une tendance actuelle du secteur de l'énergie est la décentralisation de la production. Ceci est dû à la fois à la déréglementation des marchés de l'énergie et à l'essor des technologies, qui permettent de plus en plus aux acteurs locaux de devenir de petits producteurs d'énergie (solaire, éolienne, hydrogène, etc), tout en restant des consommateurs. Ces producteurs/consommateurs locaux peuvent être des usines, des exploitations agricoles et même des particuliers. Cela soulève des questions complexes pour les producteurs d'énergie traditionnels qui, bien qu'ayant perdu leur position monopolistique, restent des acteurs clés, en raison de leur contrôle sur les réseaux de base et les principales centrales hydrauliques et nucléaires. Dans le cadre des activités du Labex IMOBS3, nous participons à un projet sur la conception et le contrôle d'une micro-usine pour la production d'hydrogène. Cette micro-usine doit fournir aux véhicules autonomes une énergie électrique obtenue à l'aide de piles à combustible à l'hydrogène. Alors que la plupart des productions d'hydrogène sont généralement réalisées par des procédés d'électrolyse coûteux en énergie, les chercheurs s'appuient ici sur l'énergie solaire et la photolyse [20], [44], qui impliquent une quantité marginale d'énergie électrique externe. Selon ce paradigme, la production/consommation d'énergie devient endogène, et s'effectue selon une sorte de boucle fermée. Il induit un besoin de **synchronisation** de haut niveau entre les processus de production et de consommation. Pourtant, très peu de travaux abordent la question de la synchronisation des transports [35] et des processus de production d'énergie.

On vient de présenter le contexte et les motivations de ce travail. Dans la section qui suit, on présentera les objectifs de notre projet.

1.2 Objectifs de la thèse

Au cours de ce travail de recherche, on s'est focalisé sur le pilotage **synchronisé** de la production d'énergie et des activités de services d'une flotte de véhicules. On se focalise sur le « décisionnel », c'est-à-dire sur la planification synchrone des activités des véhicules d'une part, et de la micro-usine d'autre part. On doit répondre aux questions suivantes :

- Pour la micro-usine, quand produit-elle de façon à alimenter les véhicules selon leurs besoins énergétiques, au moindre coût ?
- Pour les véhicules (pour des raisons de simplicité, on se limite à un véhicule), quels parcours suivent-ils ? Quand se rechargent t-ils en carburant ? Et en quelles quantités ?

Nous désignons par **SMEPC** (*Simultaneous Management of Energy Production and Consumption*) notre problème. Le problème SMEPC ainsi posé est spécifique, mais il renvoie à des problématiques générales qui font elles aussi l'objet de l'étude :

- Une problématique liée à la gestion algorithmique du mécanisme de synchronisation ;
- Une problématique qui est la prise en compte d'une dimension collaborative dans les procédés algorithmique : on a différents acteurs et il faut garder une certaine flexibilité ;
- Une problématique de robustesse, liée aux incertitudes sur la production car il nous faut des stratégies adaptées en temps réel.

Ces trois grandes problématiques font parties de l'étude, et elles vont motiver certaines innovations :

- Utilisation de la programmation linéaire en nombres entiers pour modéliser notre problème ;
- Utilisation du cadre programmation dynamique (DPS : *Dynamic Programming scheme*) pour la gestion des mécanismes de synchronisation ;
- Gestion de la dimension collaborative au travers de processus DPS communicants ;
- Utilisation des techniques d'apprentissage, plus précisément du *Deep learning*, pour l'approximation des coûts.

La présente contribution porte sur la gestion synchrone d'une flotte de véhicules équipés de piles à hydrogène, qui sont nécessaires pour effectuer des tâches logistiques locales dans une zone restreinte, et d'une micro-usine chargée de produire le combustible à hydrogène qui sera périodiquement chargé par ces véhicules. Pris dans son ensemble, il comporte simultanément des caractéristiques associées à la prévision, à la sécurité (associée à l'autonomie des véhicules) et à la synchronisation : il faut faire correspondre un calendrier d'enlèvement et de livraison (*Pickup and Delivery* en Anglais) avec la planification de la production ou du stockage d'hydrogène de la micro-usine.

Ainsi, nous considérons dans cette thèse qu'on a :

- un seul véhicule utilisé pour effectuer des tâches de livraison. Ce véhicule commence son parcours avec une certaine quantité d'hydrogène, et son réservoir a une capacité limitée. Il doit retourner à la micro-usine pour faire le plein. L'ordre de visite des stations ne changera pas.
- Une micro-usine avec une certaine capacité de stockage, dont la production dépend du rayonnement solaire.

Notre objectif est de planifier simultanément les opérations de recharge du véhicule et l'activité de production/stockage de la micro-usine. Pour cela, on doit concevoir des algorithmes et modèles pour planifier la production d'hydrogène par photolyse et la tournée du véhicule. Plus précisément, la planification des opérations de recharge du véhicule consiste à calculer à quelle date le véhicule va se recharger en hydrogène et quelles sont les quantités rechargées à chaque fois. On doit minimiser la durée de la tournée et l'énergie consommée par le véhicule. La planification de l'activité de production/stockage de la micro-usine vise à calculer à quelle date la micro-usine va produire l'hydrogène dont le véhicule aura besoin pour réaliser sa tournée. On doit minimiser les coûts de production en satisfaisant les recharges du véhicule le plus rapidement possible.

Le problème **SMEPC** est un nouveau problème. De plus, un point crucial est de concevoir des instances qui seront utiles pour tester numériquement chaque modèle conçu. On explorera aussi la complexité théorique des méthodes de résolution du problème **SMEPC**.

Nous allons dans la section suivante présenter brièvement le contenu de chaque chapitre de ce manuscrit.

1.3 Plan de la thèse

Ce rapport est composé de six principaux chapitres. On rappelle que dans notre travail, le parcours du véhicule est fixé.

Le **chapitre 2** fait un état de l'art du problème **SMEPC** (*Simultaneous Management of Energy Production and Consumption*). On analyse le positionnement de notre problème par rapport aux problèmes de même nature de la littérature. La planification de tournées avec prise en compte de l'énergie pour alimenter les véhicules a été étudiée sous plusieurs formes : problème GVRP (*Green Vehicle Routing Problem*), problème RVRP (*Recharging Vehicle Routing Problem*), problème EVRP (*Electric Vehicle Routing Problem*), problème HVRP (*Hydrogen Vehicle Routing Problem*), problème PRP (*Pollution Routing Problem*). Le problème de planification de la production a été beaucoup étudié et on indique quelques références qui résolvent ce problème. De plus, on s'intéresse aussi aux méthodes de résolution des problèmes de synchronisation qui se rapproche du problème **SMEPC**. On finit le chapitre 2 en présentant le problème IRP (*Inventory Routing Problem*).

Le **chapitre 3** fait un état de l'art des méthodes utilisées pour résoudre le problème **SMEPC**. Après avoir indiqué brièvement ce qu'est un programme en nombres entiers mixtes, on s'intéresse aux méthodes de résolution par schéma de programmation dynamique ainsi qu'aux méthodes de résolution par heuristique gloutonne. On aborde aussi les méthodes d'apprentissage par réseaux de neurones. Ce chapitre est consacré exclusivement à expliquer ces méthodes et à faire une revue de la littérature des nouvelles techniques qui ont été mises au point dans ces secteurs.

Le **chapitre 4** explique ce qu'est le problème **SMEPC**. On présente le modèle mathématique du problème **SMEPC**, et une formulation linéaire mixte MILP (Mixed Integer Linear Programming). Dans ce chapitre nous démontrons aussi que le modèle résultant est NP-Hard. On présente les procédés de génération des deux paquets instances qu'on nomme INST_VAR et INST_CTE. Ce dernier paquet d'instances a été généré à l'aide d'un procédé conçu en 2022 dans le cadre d'un projet de Master réalisé par Alejandro Olivas Gonzalez. Nous finissons ce chapitre en présentant les résultats des expérimentations numériques de nos modèles implémentés sous CPLEX.

Le **chapitre 5** propose un schéma de programmation dynamique (DPS) désigné par **DPS_SMEPC**, proche du schéma de programmation dynamique de [13], [29]. Ce qui nous permet d'énoncer un

schéma d'approximation (*Polynomial Time Approximation Scheme*). Cependant, il s'est avéré que, durant les tests, le DPS tend à impliquer un nombre excessivement élevé d'états. Nous introduisons alors des mécanismes de filtrage : des mécanismes basés sur la logique, des mécanismes basés sur l'anticipation des incohérences, des mécanismes basés sur la borne supérieure, des mécanismes basés sur le pré-calcul d'une borne inférieure et d'une solution initiale réalisable, et des mécanismes heuristiques, basés sur l'utilisation de mécanismes de dominance faible. Une partie de l'étude est consacrée à l'évaluation de la puissance de filtrage de ces procédés. L'expérimentation numérique vise principalement à évaluer pour le DPS l'impact des différents mécanismes de filtrage et la qualité de l'heuristique de calcul de la borne supérieure.

Le **chapitre 6** se focalise sur la définition d'une heuristique qui résout le problème **SMEPC**. On désigne par **Pipe-line VD_PM** cette heuristique. Les temps de calcul du **DPS_SMEPC** augmentent considérablement au fur à mesure que l'instance grossit. De plus, l'algorithme **DPS_SMEPC** ne fait pas ressortir la dimension collaborative qui existe entre les sous-problèmes de **SMEPC**. C'est pour ces raisons qu'on a mis en place une heuristique dont l'objectif est de traiter les instances d'une très grande taille et de faire communiquer deux modules qui traitent chaque sous-problème de **SMEPC**. Cette heuristique est basée sur un découpage du problème **SMEPC** en deux problèmes que l'on nomme le **problème Vehicle-Driver (VD)** et le **problème Production-Manager (PM)** traités par des schémas de programmation dynamique de façon disjointe dans un premier temps, puis dans un second temps de façon à faire communiquer ces deux problèmes afin d'obtenir une solution du problème **SMEPC**. Le schéma de programmation dynamique qui résout le problème **PM** implique un grand nombre d'états, pour cela on va mettre en place des mécanismes de filtrage similaires à ceux du chapitre 5 pour limiter ce nombre. On réalise les expérimentations numériques, qui visent principalement à évaluer l'impact des différents mécanismes de filtrage et la qualité de l'heuristique Pipe-line **VD_PM**.

Le **chapitre 7** présente des estimateurs de coûts de solutions d'instances du problème **SMEPC**. Pour faire l'approximation des coûts, on utilise les procédés d'apprentissage appelés réseaux de neurones. On construit des réseaux de neurones basés sur l'architecture Perceptron Multicouche. Le nombre de poids des réseaux de neurones étant élevé, on cherche à les diminuer. Pour cela, on va définir un ensemble d'indicateurs qui permettra de diminuer drastiquement les poids de nos réseaux de neurones.

Pour terminer, le **chapitre 8** revient sur l'ensemble des travaux réalisés, notamment l'approche MILP, l'approche programmation dynamique, l'approche heuristique et les résultats expérimentaux obtenus. Un ensemble de perspectives de recherche sont évoquées dans le cadre d'une poursuite de ces travaux. Le premier aspect concerne la planification de la tournée du véhicule, tournée que nous avons supposée fixe dans notre travail. Le second envisage l'incertitude liée à la production d'hydrogène du problème **SMEPC**.

CHAPITRE 2

ÉTAT DE L'ART DU PROBLÈME SMEPC

Sommaire

2.1	Introduction	29
2.2	Recherche Opérationnelle et Énergie	29
2.2.1	Planification des tournées de véhicules et alimentation des véhicules	30
	GVRP : <i>Green Vehicle Routing Problem</i>	30
	PRP : <i>Pollution-Routing Problem</i>	33
	EVRP : <i>Electric Vehicle Routing Problem</i>	35
	HVRP : <i>Hydrogen Vehicle Routing Problem</i>	37
	RVRP : <i>Recharging Vehicle Routing Problem</i>	38
2.2.2	Planification de la production	38
2.3	Problématique générale de synchronisation	40
2.4	Conclusion	42

2.1 Introduction

Le problème de tournées de véhicules (en anglais *Vehicle Routing Problem* et en abrégé VRP) est une classe de problème en recherche opérationnelle et en optimisation combinatoire. Cette classe de problème a été définie pour la première fois par Dantzig et Ramser en 1959 [28]. Les auteurs proposent une étude concernant la définition d'une flotte de véhicules qui fonctionnent au gasoil et qui fait des livraisons entre un client particulier et les autres clients. Les auteurs présentent un programme linéaire qui modélise ce problème dont l'objectif est de minimiser la distance parcourue par les véhicules. Dans cette classe de problème dans la plupart des cas on calcule les tournées d'un ensemble de véhicules afin de livrer une liste de clients désignés comme des stations. L'objectif est de servir ces clients en minimisant un ou plusieurs critères liés au coût de livraison des paquets. Ce problème est une extension classique du problème du voyageur de commerce. De nombreuses variantes du problème de tournées de véhicules sont survenues au fil des années, à l'instar du problème de tournées de véhicules avec contraintes de capacité (C-VRP) où les véhicules ont une capacité limitée (quantité, taille, poids, etc.), du problème de tournées de véhicules avec fenêtres de temps (VRP-TW) où pour chaque client on impose une fenêtre de temps dans laquelle la livraison doit être effectuée, ou encore du problème de tournées de véhicules avec collecte et livraison (PD-VRP) où un certain nombre de marchandises doivent être déplacées de sites de collecte vers des sites de livraisons, etc. Ce chapitre est consacré à l'état de l'art du problème **SMEPC**.

2.2 Recherche Opérationnelle et Énergie

L'émergence des sources d'énergies renouvelables (photovoltaïque, éolienne, à base d'hydrogène ou de biomasse : (voir [19], [20], [44], [65])), qui visent à remplacer les gaz émetteurs de CO₂ par de l'énergie électrique propre, ainsi que la rareté actuelle des infrastructures de charge/recharge connexes, ont suscité au cours des dernières décennies l'intérêt des chercheurs. La plupart des contributions ont porté sur les véhicules électriques ou hybrides nécessaires pour minimiser leur consommation d'énergie (problème de minimisation de la pollution et problème des véhicules hybrides : (voir [36], [40], [51], [54], [59], [61], [66])). Des modèles connexes prennent en compte les transactions de recharge soumises à des fenêtres temporelles ou à des contraintes d'accès partagé (voir [70], [85], [88], [94]). Certains auteurs ont également abordé la question de la programmation d'un processus industriel (voir [90], [7], [79], [82], [29], [76]) tout en tenant compte des variations temporelles des coûts énergétiques, des restrictions d'accès et des préoccupations environnementales. Bien que la plupart des contributions aient été associées à des applications, il convient de mentionner l'existence de plusieurs contributions théoriques qui traitent des questions de complexité et d'approximation, pour des modèles qui mettent en jeu le coût de l'inactivité et l'impact des dépendances temporelles (voir [90], [7], [82], [83], [29], [49]). D'autre part, les principaux producteurs d'énergie mènent depuis longtemps des études systématiques sur la manière de planifier la production d'énergie (gaz, électricité, gestion des barrages ou des centrales nucléaires), selon une approche à grande échelle, c'est-à-dire dans le but de répondre à des demandes agrégées incertaines à grande échelle (voir [17], [74], [75], [80], [16]).

Dans cette section, on présentera d'abord les problèmes de tournées de véhicules qui mettent l'accent sur la gestion de l'énergie c'est-à-dire qui tiennent compte de l'alimentation des véhicules lors de la planification des tournées de véhicules. Ensuite, on fera un état de l'art des problèmes

de production et enfin, on présentera des problèmes de la littérature qui jumellent planification de tournées de véhicules et production d'énergie à l'aide d'un mécanisme de synchronisation.

2.2.1 Planification des tournées de véhicules et alimentation des véhicules

Plusieurs auteurs ont traité des problèmes de tournées de véhicules qui mettent l'accent sur la gestion de l'énergie qui alimente les véhicules au cours des tournées. Ces problèmes appartenant à la classe de problème VRP sont appelés Electric VRP (EVRP), Green VRP (GVRP), Hydrogen VRP (HVRP), Recharging VRP (RVRP) (voir figure (2.1)). On s'intéresse à la façon dont les véhicules sont alimentés en énergie et à la façon dont on gère l'énergie contenue dans les véhicules. A cet effet, on s'intéresse à la planification des tournées de véhicules en tenant compte du fait que ces véhicules doivent se recharger en énergie pour pouvoir se déplacer. Dans la suite, on présente toutes ces classes de problèmes.

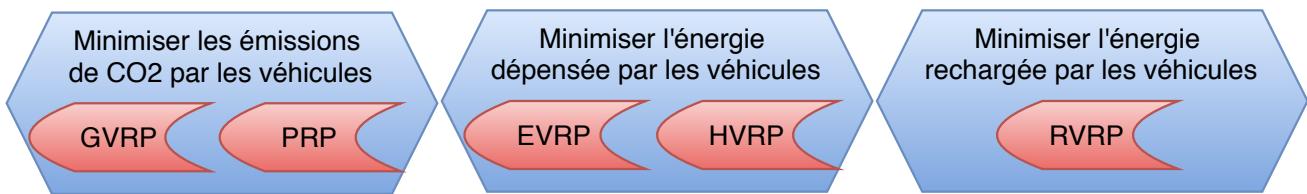


FIGURE 2.1 – Classes de problèmes qui incluent tournée de véhicules et gestion de leurs alimentations.

GVRP : *Green Vehicle Routing Problem*

Le terme « logistique verte » définit un sujet qui a été intensément étudié (voir [3] [6] [43] [51] [54] [55] [59] [70] [78] [85] [98] [105]) et a attiré l'attention de nombreux professionnels de la recherche opérationnelle. La classe de problème *Green Vehicle Routing Problem* (G-VRP) appartenant à la classe de problème VRP est alors apparue. G-VRP vise à minimiser les émissions de dioxyde de carbone (CO₂) dans les systèmes logistiques grâce à une meilleure planification des livraisons effectuées par une flotte de véhicules. G-VRP a été énoncé pour la première fois par Sevgi Erdogan et Elise Miller-Hooks en 2012 dans l'article [54].

Le G-VRP est défini dans l'article [55] sur un graphe complet orienté $G = (N, A)$, où $N = "0" \cup N_c \cup F$ est l'ensemble des nœuds et $A = \{(n_i, n_j) : n_i, n_j \in N\}$ est l'ensemble des arcs qui relient les nœuds dans N . La signification de l'ensemble des nœuds N est la suivante :

- "0" correspond au dépôt;
- $N_c = \{n_1, n_2, \dots, n_c\}$ est l'ensemble des nœuds clients et le nombre de client maximal est c ;
- $F = \{n_{c+1}, n_{c+2}, \dots, n_{c+s}\}$ est l'ensemble des nœuds de recharge (aussi appelé station-service) et le nombre de stations de recharge maximal est s .

Un nombre illimité de véhicules homogènes est disponible au dépôt pour servir les clients avec une capacité de réservoir de carburant Q (litres) et un taux de consommation de carburant r (litres par kilomètre). Chaque véhicule se déplace sur le graphe G à une vitesse constante sp (kilomètres par heure). Chaque arc $(n_i, n_j) \in A$ est associé à une distance non négative d_{ij} , un temps de parcours t_{ij} associé à la distance ($t_{ij} = d_{ij}/sp$). Chaque nœud est associé à un temps de service p_i , qui représente le service. Le dépôt peut servir de station-service et toutes les stations de carburant ont une capacité illimitée. Il n'y a pas de limite au nombre d'arrêts pour le ravitaillement en carburant et le réservoir du véhicule est supposé être plein après avoir quitté une station-service. Il existe un temps maximal

(T_{max}) qui fixe la durée d'une journée de travail et le problème est de déterminer les tournées des véhicules de manière à minimiser le coût total, en se basant sur les hypothèses suivantes : chaque véhicule est utilisé pour un itinéraire au maximum ; chaque itinéraire commence et se termine au dépôt ; le niveau de carburant dans le réservoir du véhicule doit être supérieur ou égal à la consommation de carburant entre deux nœuds quelconques ; la quantité de carburant dans le réservoir d'un véhicule est suffisante pour pouvoir circuler entre deux nœuds quelconques ; la durée de l'itinéraire attribué à un véhicule ne peut pas dépasser la durée maximale.

Dans l'article [54], Sevgi Erdogan et Elise Miller-Hooks définissent le problème G-VRP. Dans leur définition, on a plusieurs stations de recharge et elles sont distinctes du dépôt. Ils proposent deux heuristiques : l'une basée sur l'algorithme de Clark et Wright et l'autre basée sur l'algorithme *Density Based Clustering* (DBCA). Ils comparent leurs résultats à leur modèle linéaire du problème de G-VRP. Il en résulte que l'algorithme DBCA est meilleur que l'algorithme de Clark et Wright car il fournit de meilleures solutions. Les auteurs montrent que la faisabilité du problème dépend fortement de l'emplacement des stations et des stations de recharge. On minimise ici la distance parcourue par les véhicules.

Dans l'article [70], Marcos Raylan Sousa Matos et al. traite en 2018 le problème Green Vehicle Routing Problem (G-VRP). Dans ce document, le GVRSP prend en compte les véhicules hétérogènes, la congestion, et les contraintes de capacité. La livraison peut être fractionnée. Ils proposent un algorithme hybride qui combine une métaheuristique Iterated Local Search (ILS) et la procédure Random Variable Neighborhood Descent (RVND). En outre, ils utilisent un ensemble de couverture exact (SC), qui détermine la meilleure combinaison de tournées générées pendant l'exécution de l'ILS. Leur algorithme a permis de réduire le coût du CO₂ dans les 140 instances testées par rapport aux résultats du GVRSP sans livraison fractionnée présentée dans [96], tout en ayant un délai compétitif.

Dans l'article [85], Martin Sachenbacher et al. abordent le problème G-VRP de façon théorique en considérant les stations comme des sommets d'un graphe noté G , les routes qui relient les stations comme des arcs de G et les tournées comme des chemins. Chaque arc est étiqueté par la quantité d'électricité qu'il faut à un véhicule électrique pour le parcourir. Ils calculent le plus court chemin entre deux sommets u et v du graphe G en utilisant plusieurs algorithmes connus. Plus précisément, ils comparent en 2011 les résultats de l'algorithme Dijkstra et de l'algorithme A^* . Le plus court chemin entre deux sommets u et v est défini ici comme le chemin qui nécessite la plus petite quantité d'énergie au véhicule électrique pour se déplacer de u à v . Il en ressort l'algorithme A^* fournit de meilleurs résultats car il est le plus rapide.

Dans l'article [51], Imdat Kara et al. définit en 2007 un nouveau problème qu'ils appellent *Energy Minimizing Vehicle Routing Problem* (EM-VRP). Ils définissent le problème C-VRP sur un graphe $G = (V, A)$ où $V = 0, 1, 2, \dots, n$ est l'ensemble des nœuds (sommets), 0 est le dépôt (origine, ville d'origine), et les autres nœuds sont des clients. L'ensemble $A = \{(i, j) : i, j \in V, i \neq j\}$ est un ensemble d'arcs (ou d'arêtes). Chaque client $i \in V - \{0\}$ est associé à une demande entière et positive q_i et à chaque arc (i, j) est associé un coût de déplacement c_{ij} (qui peut être symétrique, asymétrique, déterministe, aléatoire, etc.). On a m véhicules de capacité identique Q . Le C-VRP consiste à déterminer un ensemble de m tournées de véhicules avec un coût minimum de telle sorte que : chaque tournée commence et se termine au dépôt, chaque client est visité par une unique tournée, et la demande de chaque tournée ne dépasse pas la capacité des véhicules Q . La C-VRP a été définie pour la première fois par Dantzig et Ramser en 1959 [28]. Dans cette étude, les auteurs ont utilisé la distance parcourue comme substitut de la fonction de coût. Depuis lors, le coût du voyage du nœud i au nœud j , c'est-à-dire c_{ij} , a

généralement été considéré comme la distance entre ces nœuds. Les auteurs définissent une nouvelle fonction de calcul du coût pour le problème Capacited Vehicle Routing Problem (C-VRP) qui comprend la distance parcourue par le véhicule et la charge du véhicule sur chaque arc. Lorsqu'ils comparent les résultats obtenus sur CPLEX de C-VRP et EM-VRP, ils constatent que EM-VRP calcule des tournées qui minimisent l'énergie totale consommée par les véhicules.

Dans l'article [55], Çağrı Koç et Ismail Karaoglan proposent en 2016 une méta-heuristique basée sur le recuit simulé pour calculer une borne supérieure du problème G-VRP. La borne inférieure est calculée avec l'algorithme *branch-and-cut*. Les résultats des calculs montrent que 22 des 40 cas avec 20 clients peuvent être résolus de manière optimale en un temps de calcul raisonnable.

Dans l'article [6], Juho Andelmin et Enrico Bartolini proposent en 2017 un algorithme exact pour résoudre G-VRP. Ils modélisent le G-VRP comme un problème de partitionnement d'ensemble dans lequel les colonnes représentent les tournées possibles correspondant à des circuits simples dans un multigraph. Chaque nœud du multigraph représente un client et chaque arc entre deux clients représente un chemin à travers un ensemble de stations de recharge visitées par un véhicule lorsqu'il voyage directement entre les deux clients. Ils renforcent la formulation du partitionnement de l'ensemble en ajoutant des inégalités valides, y compris des coupes de parcours, et ils décrivent une méthode pour les séparer. Ils fournissent des résultats sur des instances montrant que l'algorithme peut résoudre de manière optimale des instances ayant jusqu'à environ 110 clients.

Dans l'article [59], Yiyo Kuo propose en 2010 un algorithme basé sur le recuit simulé pour résoudre le problème *time-dependent vehicle routing problem* (TDVRP). L'auteur calcule ici des tournées qui minimisent la consommation de carburant par les véhicules. Une évaluation expérimentale de la méthode proposée est effectuée. Les résultats montrent que la méthode proposée permet une amélioration de 24,61 % de la consommation de carburant par rapport à la méthode basée sur la minimisation du temps de transport et une amélioration de 22,69 % par rapport à la méthode basée sur la minimisation des distances de transport. De plus, un modèle est proposé pour calculer la consommation totale de carburant pour le problème TDVRP. Dans le modèle, la consommation de carburant prend en considération le poids du chargement des véhicules.

Dans l'article [78], Nur Normasari et al. proposent en 2019 un algorithme basé sur le recuit simulé pour résoudre le problème *Capacitated Green Vehicle Routing Problem* (CG-VRP), qui est une extension du problème GVRP, caractérisé par l'objectif d'harmoniser les coûts environnementaux et économiques en mettant en œuvre des tournées efficaces. Ils formulent le modèle mathématique du problème et mettent en place un MIP. L'objectif du CGVRP est de minimiser la distance totale parcourue par un véhicule à carburant alternatif. Les résultats de l'expérience numérique montrent que l'algorithme est capable d'obtenir de bonnes solutions du CG-VRP dans un délai raisonnable, et l'analyse de sensibilité démontre que la distance totale dépend du nombre de clients et de l'autonomie du véhicule.

Dans l'article [3], Alizadeh Foroutan et al. proposent en 2020 un programme non-linéaire mixtes (MINLP) et deux méta-heuristiques basées sur le recuit simulé et l'algorithme génétique pour résoudre le problème G-VRP avec une flotte de véhicules hétérogènes. Il en résulte que l'algorithme basée sur le recuit simulé fournit de meilleures solutions et l'algorithme génétique fournit des solutions plus rapidement sur des instances de grandes tailles dont le nombre de stations est 30.

Dans l'article [98], Yiyong Xiao et al. proposent en 2019 un programme linéaire à variables entières mixtes (MILP) et une heuristique basée sur la recherche à voisinage itératif (INS) appelé *Dynamic Partial Optimization with Iterative Neighborhood Search* (DPO-INS) pour résoudre le problème *Electric Vehicle Routing Problem With Time Windows considering Energy Consumption Rate* (EVRPTW-ECR).

Dans EVRPTW-ECR, on veut planifier les tournées de véhicules en minimisant la distance parcourue, la vitesse de parcours et la quantité d'énergie électrique consommée au cours du parcours. Les résultats démontrent que le MILP résout des instances jusqu'à 25 stations et l'heuristique résout des instances jusqu'à 100 stations.

Dans l'article [43], Antonio Giallanza et Gabriella Li Puma proposent en 2020 une méta-heuristique basée sur l'algorithme génétique et un modèle de programmation par contrainte pour résoudre le problème *three-echelon fuzzy green vehicle routing problem* abrégé 3E-FGVRP. Dans ce problème on veut mettre en place un *agri-food supply chain* qui consiste à planifier le cycle suivant : la recherche de matière première, la production de marchandises, le stockage de marchandises, le transport de marchandises et la livraison de marchandises. Les auteurs font du mono-objectif en minimisant la somme de la distance parcourue par les véhicules, du coût de distribution de la marchandise et du coût d'émissions de carbone. Les auteurs réalisent une étude de cas de l'*agri-food Sicilien*.

Dans l'article [105], Ferani E. Zulvia et al. proposent en 2020 un algorithme *Many-Objective Gradient Evolution* (MOGE) basé sur l'algorithme *gradient evolution* (GE) pour résoudre le problème GVRP avec fenêtre de temps. GE consiste à parcourir un espace de solutions à l'aide de trois opérations : la mise à jour, le saut et le rafraîchissement. Les véhicules sont utilisés pour le transport de produits périssables. L'objectif est de minimiser :

- Le coût opérationnel qui est constitué du coût du carburant et du coût d'utilisation des véhicules ;
- Le coût de détérioration des produits ;
- Le coût d'émission de carbone ;
- Le coût de satisfaction des clients (respecter les fenêtres de temps attribuées à chaque client).

PRP : *Pollution-Routing Problem*

Le *Pollution-Routing Problem* (PRP) a été formulé pour la première fois en 2011 par Bektas et Laporte dans [14]. Ce problème a été abordé dans plusieurs articles (voir [30], [31], [40], [41], [56], [81], [92], [97]). C'est un problème d'optimisation qui consiste à trouver un équilibre entre coût monétaire et coût environnemental (émission de gaz à effet de serre) des entreprises de logistique qui livrent des clients avec des véhicules alimentés aux combustibles fossiles. Dans [14], les auteurs préspécifient un ensemble de vitesses que les véhicules peuvent prendre, ce qui conduit à des solutions sous-optimales car les temps de trajet sont discrétisés. Cette stratégie de discrétilisation est aussi utilisée par Franceschetti et al. [41] pour résoudre le *Time-Dependent PRP* (TD-PRP), par Demir et al. [31] pour résoudre le problème Bi-objective PRP et par Koc et al. [56] pour résoudre le problème Heterogeneous PRP.

Le problème PRP consiste à calculer les tournées de véhicules pour servir un ensemble de clients dans le but de minimiser les émissions de gaz à effet de serre. Les véhicules sont confrontés à des embouteillages qui, aux périodes de pointe, limitent considérablement leurs vitesses et entraînent une augmentation des émissions de gaz à effet de serre.

Dans la littérature, PRP a été défini de plusieurs façons en fonction du critère à optimiser. D'après [14], on peut classifier ces définitions en quatre catégories à savoir les modèles minimisant les émissions de gaz à effet de serre en optimisant :

- La charge des véhicules ;
- La vitesse des véhicules ;

- Simultanément la charge et la vitesse des véhicules ;
- La charge des véhicules tout en évitant les congestions.

Dans l'article [30], E. Demir et al. proposent en 2012 un algorithme ALNS qui résout le problème Pollution-Routing. La métaheuristique ALNS est une extension de l'heuristique Local Neighborhood Search (LNS) proposée pour la première fois par Shaw [89] en 1997, et basée sur l'idée de modifier une solution initiale au moyen d'opérateurs de destruction et de réparation. Si la nouvelle solution est meilleure que la meilleure solution actuelle, elle la remplace et sert d'entrée à l'itération suivante. La métaheuristique ALNS quant à elle a été proposée par Ropke et Pisinger [84] en 2006 pour résoudre des variantes du VRP. Plutôt que d'utiliser un grand voisinage comme dans le LNS, elle applique plusieurs opérateurs de retrait et d'insertion à une solution donnée. Les opérateurs d'insertion sont utilisés pour réparer une solution partiellement détruite en réintégrant les stations de la liste de retrait dans la solution. Ces opérateurs réintègrent les stations supprimées dans les tournées existantes lorsque la faisabilité en termes de capacité et de délais peut être maintenue, ou ils créent de nouvelles tournées. Le voisinage d'une solution est obtenue en retirant certains stations de la solution et en les réintégrant d'une autre façon.

Dans l'article [31] les auteurs proposent en 2013 un algorithme ALNS qui résout le *Bi-objective Pollution-Routing Problem* qui est une extension du problème *Pollution-Routing Problem* (PRP). Dans le *Bi-objective Pollution-Routing Problem*, on a deux objectifs : l'un est de minimiser l'énergie consommé par les véhicules et l'autre est de minimiser la distance parcourue par les véhicules.

Dans l'article [40], Anna Franceschetti et al. proposent en 2017 une métaheuristique pour le problème *Time-Dependent Pollution-Routing*. Ici, on minimise aussi le coût du salaire du conducteur. Leur algorithme est basé sur une heuristique ALNS et utilise de nouveaux opérateurs de retrait et d'insertion qui améliorent considérablement la qualité de la solution. Une procédure d'optimisation des heures de départ et de la vitesse des véhicules, mise au point antérieurement, est utilisée comme sous-programme pour optimiser les heures de départ et la vitesse des véhicules. Les auteurs comparent les résultats obtenus par l'ALNS avec les résultats de CPLEX.

Dans l'article [97], Yiyong et al. proposent en 2020 un *ϵ -accurate mixed integer linear program* pour résoudre le problème *continuous PRP*. Ici les variables à optimiser peuvent prendre tout un continuum de valeurs. Le coefficient ϵ est le pourcentage d'erreur maximal admissible dans la linéarisation des relations entre le temps, la vitesse et la consommation de carburant. Les auteurs minimisent le temps de trajet des véhicules et la quantité de carburant consommée par les véhicules. Les expérimentations ont montré que le modèle ϵ -CPRP pouvait donner de bonnes solutions. De plus, de nombreuses instances *UK_PRP* ont été mises à jour avec de meilleures solutions.

Dans l'article [81], Rui et al. définissent en 2020 un nouveau variant du problème PRP qu'ils appellent *carbon pricing initiatives-based bi-level pollution routing problem* (CPIBPRP). Les gouvernements ont fixé des prix du carbone afin que les entreprises diminuent leurs émissions de carbone. Autrement dit, plus une entreprise produira du carbone, plus la taxe qu'elle devra payer sera élevée. CPIBPRP cherche à programmer les tournées de véhicules et on a deux objectifs à atteindre. Premièrement, on minimise les émissions de carbone et, deuxièmement, on doit respecter les limites imposées par le gouvernement concernant la quantité de carbone à produire. Les auteurs définissent d'abord un modèle à deux niveaux pour CPIBPRP. Ensuite, pour résoudre ce problème les auteurs développent un algorithme interactif intégrant l'optimisation par essaims de particules contrôlée par la logique floue et une heuristique ALNS modifiée. Ils le nomment *algorithm integrating fuzzy logic controlled particle swarm optimization (FLCPSO)* and the modified ALNS heuristic (IFLCPSO-MALNS). Les résul-

tats montrent que l'application des initiatives de tarification du carbone peut réduire les émissions de carbone en augmentant relativement peu le coût total des entreprises de transports de marchandises.

Dans l'article [92], Erfan et al. définissent en 2020 un variant du problème PRP qu'ils appellent *Pollution-Routing Problem with Cross-dock Selection* (PRP-CDS). Le cross docking est une méthode d'expédition très utilisée pour simultanément réduire les coûts de stockage, transporter la marchandise des fournisseurs aux clients le plus rapidement possible et minimiser les dommages causés à la marchandise. Cette méthode se déroule en trois étapes :

1. On reçoit la marchandise dans les entrepôts de cross docking ;
2. On décharge la marchandise, on la vérifie et on la trie en fonction de sa destination ;
3. On expédie la marchandise aux clients le plus rapidement possible.

PRP-CDS consiste à planifier le trajet des véhicules en visant deux objectifs :

- Premièrement, minimiser le coût de pollution et le coût de transport des marchandises. Ici les conditions du trafic et la vitesse des véhicules déterminent la consommation de carburant et la pollution ;
- Deuxièmement, maximiser la satisfaction des clients liée à la quantité moyenne de marchandises transportées vers les entrepôts de cross docking.

Pour résoudre le problème PRP-CDS, les auteurs développent une méthode exacte *Bi-Objective Mixed Integer Linear Programming* (BOMILP) et deux métaheuristiques dont l'une est basée sur le *Multi-Objective Simulated-annealing Algorithm* (MOSA) et l'autre est basée sur le *Non-dominated Sorting Genetic Algorithm II* (NSGA-II). Après la phase expérimentale, il en ressort que les algorithmes proposés fournissent de bonnes solutions et que c'est l'algorithme NSGA-II qui est le plus efficace.

EVRP : Electric Vehicle Routing Problem

Les véhicules électriques sont une nouvelle génération de véhicules qui fonctionnent à l'aide de batteries. La durée de recharge d'une batterie est longue, raison pour laquelle les modèles et algorithmes de VRP classique ne sont pas adaptés pour planifier les trajets de ces véhicules, d'où l'apparition de la classe de problèmes Electric VRP (E-VRP). De nombreux auteurs ont conçu des modèles et algorithmes pour traiter les problèmes E-VRP (voir [12], [22], [36], [38], [50], [60], [68], [88], [94], [101], [102]).

Dans l'article [102], Shuai Zhang et al. proposent en 2018 une méta-heuristique basée sur l'algorithme Colonie de Fourmis (dont les perturbations sont effectuées par Iterated Local Search) pour résoudre E-VRP. On a ici plusieurs stations de recharge dont le dépôt ne fait pas partie. Ils montrent qu'utiliser l'énergie consommée par les véhicules comme fonction objectif donnent de meilleures solutions que de considérer la distance comme fonction objectif. Ils ont implémenté un algorithme *Adaptative Local Neighborhood Search* (ALNS) et un modèle linéaire qu'ils comparent avec la métaheuristique. Ils en ressort que la méta-heuristique est meilleure que l'algorithme ALNS car il fournit de meilleures solutions plus rapidement.

Dans l'article [36], Çağrı Koç et al. veulent calculer à quels endroits seront placés les stations de recharge et quelle technologie (vitesse de recharge rapide, moyenne ou lente) de recharge devra être affectée à chaque station. Ce problème est appelé dans la littérature *E-VRP with shared charging stations* (E-VRP-SCS). Ici, les stations de recharge et le dépôt sont distincts. Pour résoudre le problème E-VRP-SCS, ils proposent en 2018 une heuristique ALNS comprenant des étapes d'intensification et

de diversification de la solution réalisées par des opérateurs d'insertion et de suppression de stations. La fonction coût est constituée du coût de la tournée et du coût de l'ouverture des stations. Ils ont comparé l'heuristique ALNS à un modèle linéaire MIP.

Dans l'article [88], Michael Schneider et al. traitent en 2014 le problème *Electric Vehicle Routing Problem with Time Windows* (E-VRPTW) en tenant compte des recharges en électricité des véhicules. Chaque station a une fenêtre temporelle et un temps de service. Le service ne peut pas commencer avant la date de début de la fenêtre temporelle, ce qui peut entraîner un temps d'attente, et ne peut pas commencer après la date de fin de la fenêtre temporelle. Pour résoudre le problème E-VRPTW, ils conçoivent une heuristique basée sur une combinaison de l'algorithme de recherche à voisinage variable (*Variable Neighborhood Search*) et de la recherche tabou (*Tabou Search* en anglais). Ils démontrent l'efficacité de cette heuristique en la comparant avec une formulation linéaire implémenté sur CPLEX.

Dans l'article [68], Ji et al. définissent en 2020 un nouveau problème appelé *Time-dependent Electric Vehicle Routing Problem* (TDEVRP). L'objectif de ce problème est de planifier le trajet des véhicules électriques en tenant compte de périodes de congestion qui correspondent à des heures de pointe fixées le matin et le soir chaque jour. On tient aussi compte des décisions de recharge des véhicules qui ont lieu aux stations. On tient aussi compte ici de la quantité d'énergie consommée par les véhicules : celle-ci est calculée grâce aux poids préalablement fixés sur les routes, à la vitesse des véhicules et à la distance des routes. Pour résoudre le problème TDEVRP, ils définissent un programme linéaire en nombres entiers et un algorithme basé sur la recherche à voisinage variable. Cet algorithme fournit d'excellents résultats pour des instances de petites tailles (pas plus de 15 stations) et il fournit un gap en moyenne de 2,38% pour des instances de grandes tailles.

Dans l'article [22], Rui et al. proposent en 2019 un modélisation mathématique à deux niveaux (en anglais bi-level programming problem) du problème E-VRP-SCS. Ce modèle calcule la localisation et la capacité des stations de recharge pour des véhicules électriques. La programmation mathématique à deux niveaux résout des problèmes dont la structure hiérarchique est constituée de deux unités de prise de décision. Ici, on modélise la situation de plusieurs décideurs qui veulent réaliser leurs objectifs respectifs mais ne peuvent le faire indépendamment les uns des autres. Le problème de niveau supérieur minimise le coût de construction des installations de recharge et le coût de transport des marchandises. Le problème de niveau inférieur capture les réponses des conducteurs des véhicules aux décisions du niveau supérieur. Au cours des expérimentations du modèle, les auteurs évaluent les performances du modèle, la robustesse des solutions et l'évolutivité des solutions sur de grands réseaux.

Dans l'article [38], Rasoul et al. proposent en 2020 un framework basé sur la théorie des jeux pour résoudre le problème de routage intelligent des véhicules électriques pour l'équilibrage des charges dans un réseau intelligent. Pour cela, ils tiennent compte des embouteillages et du temps d'attente des véhicules aux stations de recharge. Le problème EVRP est formulé ici comme un jeu non coopératif répété. Si on a un ensemble de joueurs qui joue à un jeu dans lequel chacun cherche à optimiser ses gains, on aura un équilibre de Nash si chaque joueur a une stratégie tel qu'aucun ne peut obtenir un gain supplémentaire en changeant unilatéralement de stratégie. L'équilibre de Nash renvoie à un critère d'absence de regret. Il est montré dans cet article que tout équilibre de Nash obtenu améliore sensiblement l'équilibrage des charges sur le réseau.

La vulgarisation du télétravail a fait naître de nouvelles problématiques. Certains propriétaires de véhicules électriques aimeraient partager leurs véhicules de manière rentable tout en s'assurant

des heures de travail suffisantes sur un poste de travail à domicile. Dans l'article [60], Kexing et al. proposent en 2020 un modèle permettant à ces propriétaires de pouvoir faire une planification optimale en minimisant simultanément le coût de recharge de leurs véhicules et le temps de livraison des clients. Le véhicule a trois états possibles : soit il est en charge, soit il est sur le parkings, soit il est entrain de transporter des clients. Les actions du propriétaire permettent de faire passer le véhicule d'un état à un autre. Les résultats numériques obtenus avec CPLEX démontrent l'efficacité de ce modèle.

Dans l'article [50], Surendra et al. proposent en 2020 un modèle pour le problème E-VRP with Non-Linear charging and Load-Dependent discharging abrégé E-VRP-NL-LD. Dans ce problème, on a une flotte illimitée de véhicules électriques homogènes dont la capacité de recharge est fixée à Q kWh. Cette flotte est chargée de réaliser des tâches de livraison à un ensemble de clients. Pour pouvoir se déplacer, au cours de leurs parcours, ces véhicules vont se recharger en électricité dans des stations de recharge préalablement fixées. Les véhicules commencent leurs tournées dans une station particulière appelée ici dépôt. Les véhicules ont un temps maximal T_{max} pour finir leurs tournées. On veut minimiser la durée des trajets qui inclut aussi la durée des recharges. Les auteurs proposent aussi un algorithme ALNS pour résoudre ce problème. Les expérimentations numériques montrent que l'ALNS améliore les algorithmes existants pour 63% des instances et obtient la solution optimale pour 31% des instances.

Dans l'article [101], Shuai et al. proposent en 2020 un modèle d'optimisation floue du problème *electric vehicle routing problem with time windows and recharging stations* abrégé EVRPTW. Un algorithme ALNS y est aussi proposé pour résoudre ce modèle.

Dans l'article [94], Rashid Waraich et al. proposent en 2014 un modèle de simulation à événements discrets multi-agents afin d'obtenir un modèle de la dépendance temporelle des demandes et des coûts par rapport à une flotte de véhicules électriques.

HVRP : Hydrogen Vehicle Routing Problem

Dans l'article [69], Simona Mancini propose en 2017 un *Mixed Integer Linear Programming* du problème *Hydrogen Vehicle Routing Problem*. Afin de résoudre ce problème, elle propose aussi une méta-heuristique basée sur la recherche à voisinage large. Le problème HVRP consiste à visiter un ensemble de clients en partant d'un dépôt. La flotte disponible est composée d'un nombre fixe de véhicules identiques. Puisque les tournées multiples ne sont pas autorisées, au plus un nombre de tournées correspondant au nombre de véhicules peuvent être planifiées. Chaque véhicule doit commencer et terminer sa tournée au dépôt. Chaque client est caractérisé par un temps de service donné. Chaque paire de clients est associée à un temps de déplacement et à une distance. Les vitesses de déplacement sont supposées être constantes sur une route. En outre, aucune limite n'est fixée sur le nombre d'arrêts qui peuvent être effectués pour la recharge, mais chaque station de recharge ne peut être visitée qu'une seule fois. Lorsqu'une recharge est entreprise, on suppose que le réservoir est rempli à pleine capacité. Les tournées commencent et se terminent au dépôt et doivent contenir des visites aux stations de recharge, si elles sont nécessaires. Chaque trajet peut être parcouru en utilisant uniquement le moteur électrique ou le carburant. Un coût de pénalité supplémentaire est ajouté lorsque le carburant est utilisé afin de dissuader l'utilisation de ce mode et de promouvoir l'utilisation du moteur électrique. Une limite à la durée des tournées est imposée. De nombreux articles traitent ce problème (voir [12], [61], [69], [100], [104]).

Dans l'article [12], Sina et al. proposent en 2020 un algorithme exact de *branch-and-price* et une heuristique pour résoudre le problème *Plugin Hybrid Electric Vehicle* abrégé PHEV. Dans ce problème,

on a un ensemble de véhicules qui fonctionnent avec de l'électricité et de l'essence. Ces véhicules doivent réaliser des tâches chez des clients. On veut minimiser la consommation totale d'énergie en utilisant les deux sources d'énergie de façon optimale. Les auteurs montrent que l'heuristique devient plus rapide lorsque la capacité de la batterie est plus importante. Ils présentent aussi une étude de cas sur la ville de Toronto et montrent que les véhicules utilisent de l'électricité dans les zones congestionnées du centre-ville et de l'essence dans les autres zones.

Dans l'article [61], Antti Lajunen fait en 2014 des simulations afin de réaliser une analyse coût-bénéfice. Cette analyse consiste à étudier la consommation d'énergie et les émissions de CO₂ de 4 véhicules hybrides et un véhicule électrique. De cette étude il en ressort que les coûts d'investissement et de stockage sont les plus importants. De plus, ces types de véhicules permettent de réduire les émissions de CO₂ et la consommation d'énergie.

Dans l'article [100], Vincent Yu, et al. définissent en 2016 le problème Hybrid VRP. Ils donnent une formulation mathématique de ce problème et ils proposent une heuristique basée sur le recuit simulé. Ils montrent que l'heuristique proposée fournit une solution au problème HVRP.

Dans l'article [104], Lu Zhen et al. proposent en 2019 un nouveau variant du problème Hybrid VRP appelé HVRP avec mode de sélection. Ici on a un dispositif qui permet de selectionner la source d'énergie utilisée (l'électricité ou le carburant) lors des trajets. Les auteurs proposent une formulation mathématique de ce problème.

RVRP : Recharging Vehicle Routing Problem

Conrad et Figliozzi ont défini en 2011 un nouveau problème appelé *Recharging Vehicle Routing Problem* abrégé RVRP (voir [7], [90]) avec des fenêtres de temps, où les véhicules ayant une autonomie limitée peuvent être rechargés chez les clients pendant le processus de service. Le RVRP constitue la base de l'EVRP dans lequel les fenêtres de temps des clients sont prises en compte et le temps de recharge est fixe. Pour traiter le RVRP, un algorithme itératif de construction et d'amélioration d'itinéraire a été utilisé.

Dans la section suivante, on donne juste quelques références qui traitent le problème de planification de la production. Nous ne nous attarderons pas sur ce problème car c'est un problème qui a été largement étudié dans la littérature.

2.2.2 Planification de la production

La gestion de la production vise à planifier et contrôler la transformation des matières en produits finis. Elle implique la combinaison de ressources, parmi lesquelles les moyens matériels (les machines), les moyens humains (le personnel par qualification) et les matières (matières premières, matières consommables) dans un planning avec pour but d'assurer la fabrication du produit en qualité et en quantité définies. Dans un environnement économique devenu aussi concurrentiel que le notre, les enjeux financiers sont cruciaux. Le prix de vente des produits dépend de plus en plus de la demande du marché et reste très influencé par la concurrence. Afin de rester compétitive et surtout garantir une marge bénéficiaire convenable sur la vente de leurs produits, les entreprises industrielles ont pour principal recours la réduction du coût de production. Le champ d'action de la gestion de la production dans l'entreprise est vaste, couvre de nombreuses activités et interpelle les professionnels de différents domaines de formation.

Les contraintes rencontrées sont de divers ordres :

- Financières (produire à un coût optimal), coût des matières et consommables, coût de stockage des encours et des produits semi ouvrés, coût de gestion des magasins, coût des heures de travail supplémentaires, coût des arrêts faisant partie intégrante du coût de revient. Maîtriser ces derniers est aussi une garantie pour la commercialisation des produits finis.
- Temporelles (produire dans les délais, assurer une livraison juste à temps). Il s'agit d'éviter les ruptures de stocks, éviter le gonflage des stocks de produits finis. Car cela a une incidence directe sur la satisfaction de la clientèle (pertes de commandes) ou sur la partie du coût de revient des produits finis dûe aux coûts supplémentaires du stockage.
- Mécaniques (maintenance préventives et gestion des temps d'arrêt). Il s'agit d'anticiper sur les pannes et de prévoir des solutions alternatives en cas d'arrêt d'une machine.
- Qualité (produire avec le moins de défauts possible, le moins de déchets) : un produit de bonne qualité participe à la fidélisation de la clientèle et véhicule l'image de marque de l'entreprise.
- Planification, il s'agit d'assurer une circulation continue des flux, de détecter et de supprimer les goulets d'étranglement dans le circuit de production. Il s'agit donc aussi de définir un plan de production, de définir les gammes opératoires, d'ordonnancer les opérations, et enfin de gérer la répartition des tâches durant tout le processus de fabrication.

On se focalise principalement sur l'état de l'art des méthodes de planification de la production. Il existe plusieurs types de plans de production :

- Le plan de production à court terme peut être vu comme une version détaillée et désagrégée du plan de production à moyen terme : il désagrège les familles de produits en articles individuels et il planifie la production sur un horizon plus court en tenant compte d'une découpe plus fine du temps en sous-périodes.
- La planification à moyen terme prend place dans un cadre de décision où le portefeuille de produits et le processus de production peuvent être considérés comme des données irrévocables, fixées par la stratégie de l'entreprise. Les questions qui se posent à ce niveau de décision tactique portent donc sur l'utilisation optimale du système productif dans le but de satisfaire la demande prévisionnelle sur l'horizon du plan.
- Dans le très court terme, le département production se trouve confronté à une collection d'ordres de fabrication (OF) à exécuter. Chaque OF consiste en une liste d'opérations à effectuer, mais ne précise pas nécessairement l'ordre dans lequel ces opérations doivent être exécutées, ni l'instant auquel elles doivent être entamées, ni les postes de production auxquels elles doivent être affectées. Ordonnancer la production, c'est planifier les dates de lancement et de fin des opérations ainsi que l'affectation de ces opérations aux différents postes de production susceptibles de les exécuter. On classe généralement la fonction d'ordonnancement parmi les activités de planification à très court terme, quoique dans le cas de l'ordonnancement de projets elle puisse parfois tenir compte d'un horizon de plusieurs années.

La planification de la production est un problème qui a été longuement étudié au fil des années (Voir [1], [11], [16], [17], [21], [27], [32], [34], [37], [42], [46], [49], [58], [62], [63], [64], [72], [74], [75], [76], [80], [82], [83], [91], [93], [99], [103]).

Dans l'article [76], Haouassi Mustapha et al. programment en 2016 un processus de production industrielle soumis à un coût électrique linéaire à la pièce. Ils implémentent des modèles MILP et des algorithmes de programmation dynamique.

Dans l'article [80], Agnes Pechmann et Schöler se concentrent en 2011 sur la gestion des processus de consommation d'énergie qui impliquent des pics et des ruptures, comme dans l'industrie sidérurgique ou à l'intérieur de grands bâtiments.

2.3 Problématique générale de synchronisation

La synchronisation a été longuement étudiée dans la littérature (Voir [35], [39], [67], [2], [4], [5], [8], [9], [10], [23], [25], [26], [33], [45], [47], [52], [87]).

Dans l'article [39], Martin Fink et al. traitent en 2019 le problème *abstract vehicle routing problem with worker and vehicle synchronization* abrégé AVRPWVS. Ici, les auteurs cherchent à planifier la manutention au sol dans un aéroport. Les opérations planifiées sont par exemple le ravitaillement des avions et le déchargement des bagages. Pour réaliser ces tâches, Les employés doivent se déplacer d'un endroit à un autre dans l'aéroport avec des véhicules. Pour résoudre AVRPWVS, les auteurs développent un modèle mathématique et une heuristique basée sur le Branch and Price.

Dans l'article [35], Michael Drexel fait en 2012 un état de l'art du problème Vehicle Routing Problem with Multiple Synchronization Constraints dont l'acronyme est VRPMS. L'auteur présente d'abord une classification des différents types de synchronisation à savoir la synchronisation de charge, la synchronisation de ressources, la synchronisation d'opérations et la synchronisation de mouvements. Puis, Il analyse ensuite des questions centrales liées aux solutions exactes et heuristiques de ces problèmes. Il passe enfin en revue de manière exhaustive la littérature pertinente en ce qui concerne les applications et les approches de résolution.

Dans l'article [67], Ran LIU et al. proposent en 2018 une heuristique *Atadaptive Large Neighborhood Search* pour résoudre le problème *Vehicle Routing Problem With Time Windows and Synchronized Visits* (un problème de routage de véhicules avec des fenêtres de temps et des visites synchronisées). Des méthodes efficaces d'évaluation des solutions et de vérification de la synchronisation croisée y sont proposées. L'efficacité de l'approche y est ensuite prouvée.

Dans l'article [9], Archetti et al. définissent en 2007 les principes fondamentaux des problèmes basés sur l'*Inventory Routing Problem* (IRP) en proposant une formulation MIP conçue pour un problème de routage, et combinée avec des contraintes de gestion des stocks. Contrairement à Bell et al. en 1983, la définition des tournées est intégrée dans le MIP et un seul véhicule est disponible par période. En outre, les auteurs ajoutent un coût d'inventaire pour chaque client/fournisseur du système. Le modèle est résolu par un algorithme de Branch-And-Cut où les coupes sont consacrées à l'élimination des sous-tours.

Dans l'article [8], Archetti et al. proposent en 2012 un mélange d'une recherche Tabou et d'un MIP qui est appelé chaque fois que la recherche Tabou trouve une solution améliorée.

Ces dernières années, le cas des véhicules multiples, appelé problème d'acheminement de l'inventaire des véhicules multiples (MIRP), a reçu une attention considérable de la part de la communauté des chercheurs. Le MIRP a été résolu par une heuristique dans l'article [25] par Coehlo et al. en 2012 et résolu avec des méthodes exactes dans l'article [26] par Coehlo et Laporte en 2014. Les auteurs proposent pour ce problème un modèle mathématique et un algorithme de Branch-And-Cut. Ils ont également défini un nouvel ensemble de données (en modifiant les instances de 2007 proposées par Archetti) qui est utilisé par tous les articles sur la MIRP jusqu'à présent. L'échelle maximale des cas pouvant être résolus de manière optimale est d'environ 25-30 clients sur trois périodes et de 10-15 clients sur six périodes.

Coelho et al. généralisent en 2014 les inégalités présentées dans [9] et prouvent que l'ordre des clients pendant le processus de branchement a un impact significatif sur les temps de calcul.

Archetti et al. en 2014 comparent plusieurs MIP et en particulier prouvent qu'il est significativement plus efficace de générer dynamiquement des coupes pour éliminer les sous-tours que d'utiliser une formulation basée sur les flux inspirée de la formulation MTZ de Miller proposée en 1960. Ces résultats expliquent pourquoi la majorité des approches exactes tirent profit d'un algorithme de Branch-And-Cut.

En 2016, Desaulnier et al. [33] résolvent le MIRP par un algorithme de branch and price où une colonne modélise à la fois une tournée et des quantités de livraison.

Dernièrement, en 2018, Avella et al. [5] introduisent à la fois une famille d'inégalités basée sur l'analyse du problème IRP sur une période, et des algorithmes de séparation.

En ce qui concerne les approches métaheuristiques, les principales contributions se concentrent sur la génération initiale de solutions et les mouvements inter et intra-périodiques qui devraient favoriser la convergence et éviter le piège prématûr des minima locaux.

Dans l'article [87], Santos et al. proposent en 2016 une métaheuristique basée sur l'ILS qui tire parti de 14 mouvements de voisinage et d'un programme linéaire. Les mouvements sont intégrés dans un programme de descente à variables aléatoires (RVND) et dans un programme de recherche locale itérative (ILS).

Dans l'article [10], Archetti et al. introduisent en 2017 une métaheuristique à trois étapes qui combine une recherche tabou et des formulations de programmation mathématique. Alvarez et al. proposent en 2018 deux métaheuristiques pour résoudre le MIRP et une variante du MIRP dans laquelle le rapport entre les distances de routage et les quantités livrées est minimisé.

Au cours des dernières années, la majorité des contributions sont des méthodes exactes, et les méthodes métaheuristiques se sont concentrées en majorité sur des cas à grande échelle.

Le MIRP a été abordé en tenant compte de nombreuses généralisations, y compris, mais sans s'y limiter, les politiques de réapprovisionnement et les produits périssables. Par exemple, dans l'article [2], Adulyasak et al. en 2014 considère le problème de l'acheminement de la production qui est un MIRP intégrant les décisions de taille des lots de production. Les auteurs résolvent ce problème en générant plusieurs modèles de production, puis en optimisant un problème de MIRP pour chaque modèle à l'aide d'un schéma ALNS.

Dans l'article [4], Alvarez et al. abordent en 2019 un problème de MIRP avec des produits périssables pour lequel ils conçoivent un programme mixte (résolu par Branch-And-Cut) et une heuristique hybride.

Dans l'article [23], Chitsaz et al. introduisent en 2019 le problème d'acheminement de l'assemblage (ARP) qui consiste à planifier simultanément l'assemblage d'un produit dans une usine et les tournées des véhicules qui collectent les matériaux chez les fournisseurs pour répondre aux exigences de stock imposées par la production. Les auteurs conçoivent une méthode de décomposition en trois étapes et adaptent la méthode pour résoudre le MIRP.

Dans l'article [52], Panagiotis et al. proposent en 2020 un MIP et un algorithme à voisinage variable pour résoudre le problème qu'ils appellent Fleet-size and Mix Pollution Location-Inventory-Routing Problem with Just-in-Time replenishment policy and Capacity Planning.

Dans l'article [47], Mike Hewitt et al. proposent en 2013 un MIP et un algorithme Branch-and-price pour résoudre le *Maritime Inventory Routing Problem* dont l'acronyme est MIRP.

Dans l'article [45], Yun He et al. proposent en 2016 une formulation Mixed Integer Linear Pro-

gramming (MILP) du l'*Inventory Routing Problem With Explicit Energy Consumption*.

2.4 Conclusion

Dans la littérature on a trouvé plusieurs articles qui traitent différents aspects du problème **SMEPC**. Les tableaux (2.1) et (2.2) font un récapitulatif de l'état de l'art sur la planification des tournées de véhicules en mettant un accent sur l'alimentation de ces véhicules. Mais à notre connaissance il n'existe pas d'articles qui traitent le problème résolu dans le cadre de notre projet de recherche. Le chapitre suivant sera consacré à l'élaboration de l'état de l'art des méthodes utilisées ici pour résoudre le problème **SMEPC**.

Articles \ Caractéristiques	E-VRP	G-VRP	PRP	EM-VRP	HVRP	CG-VRP	IRP	RVRP
Archetti et al., 2007							X	
Archetti et al., 2012							X	
Coehlo et al., 2012							X	
Coehlo and Laporte, 2014							X	
Desaulnier et al., 2016							X	
Santos et al., 2016							X	
Archetti et al., 2017							X	
Alvarez et al., 2018							X	
Adulyasak et al., 2014							X	
Alvarez et al., 2019							X	
Chitsaz et al., 2019							X	
Hewitt et al., 2013							X	
Shuai Zhang et al., 2018	X							
Çagri Koç et al., 2018	X							
Michael Schneider et al., 2014	X							
Sevgi Erdogan et al., 2012		X						
Martin Sachenbacher et al., 2011		X						
Marcos Raylan S. et al., 2018		X						
Çagri Koç et al., 2016		X						
Juho Andelmin et al., 2017		X						
E. Demir et al., 2012			X					
E. Demir et al., 2013			X					
Anna Franceschetti et al., 2017			X					
Tolga Bektas et al., 2011			X					
Anna Franceschetti et al., 2013			X					
Çagri Koç et al., 2014			X					
Imdat Kara et al., 2007				X				
Yiyo Kuo, 2010				X				
Nur Normasari et al., 2019							X	
Alizadeh Foroutan et al. 2020		X						
Yiyong Xiao et al., 2019		X						
Antonio G. et Gabriella L. 2020		X						
Ferani E. Zulvia et al. 2020		X						
Yiyong et al. 2020			X					

suite à la page suivante

Articles	suite de la page précédente								
	E-VRP	G-VRP	PRP	EM-VRP	HVRP	CG-VRP	IRP	RVRP	
Ji et al. 2020	X								
Rui et al. 2020				X					
Erfan et al. 2020				X					
Rasoul et al. 2020	X								
Rui et al. 2019	X								
Kexing et al. 2020	X								
Surendra et al. 2020	X								
Shuai et al. 2020	X								
Rashid Waraich et al. 2014	X								
Simona Mancini 2017						X			
Sina et al. 2020						X			
Antti Lajunen fait en 2014						X			
Vincent Yu, et al. 2016						X			
Lu Zhen et al. 2019						X			
Eric Angel et al. proposent en 2014									X
Albers Susanne propose en 2010									X
Juho Andelmin et Enrico Bartolini 2017		X							

TABLE 2.1 – Récapitulatif de l'état de l'art du problème SMEPC.

Les sigles sont les suivants :

- Branch & Cut : B&C ;
- Branch & Price : B&P ;
- Clark et Wright : CW ;
- Itterative Local Search : ILS ;
- Random Variable Neighborhood Descent : RVND ;
- Colonies de fourmis : CF ;
- Recherche Tabou : RT ;
- Recuit Simulé : RS ;
- Recherche Locale : RL ;
- Gradient Evolution : GE ;
- Algorithme Génétique : AG ;
- Partitionnement d'ensemble : PE .

Articles	Caractéristiques															
	MIP	BC	BP	CW	ILS	ALNS	RVND	TS	RT	RS	RL	AG	MILP	CF	GE	PE
Archetti et al., 2007	X	X														
Archetti et al., 2012	X								X							
Coehlo et al., 2012	X					X										
Coehlo and Laporte, 2014		X														
Desaulnier et al., 2016	X	X	X													

suite à la page suivante

Articles	suite de la page précédente														
	MIP	RC	BP	CW	ILS	ALNS	RVND	CF	RF	RS	RZ	AS	MILP	GF	AG
Santos et al., 2016					X		X								
Archetti et al., 2017	X								X						
Alvarez et al., 2018															
Adulyasak et al., 2014							X								
Alvarez et al., 2019	X	X													
Chitsaz et al., 2019	X														
Hewitt et al., 2013			X									X			
Shuai Zhang et al., 2018						X		X		X					
Çagri Koç et al., 2018	X						X								
Michael Schneider et al., 2014								X		X					
Sevgi Erdogan et al., 2012	X			X											
Martin Sachenbacher et al., 2011															
Marcos Raylan S. et al., 2018					X			X							
Çagri Koç et al., 2016											X				
Juho Andelmin et al., 2017															X
E. Demir et al., 2012							X								
E. Demir et al., 2013							X								
Anna Franceschetti et al., 2017						X									
Imdat Kara et al., 2007											X				
Yiyo Kuo, 2010										X					
Nur Normasari et al., 2019										X					
Alizadeh Foroutan et al. 2020										X			X		X
Yiyong Xiao et al., 2019												X	X		
Antonio G. et Gabriella L. 2020															X
Ferani E. Zulvia et al. 2020															X
Ji et al. 2020					X										
Rui et al. 2020						X									
Erfan et al. 2020	X										X				
Rasoul et al. 2020	X														
Rui et al. 2019	X														
Kexing et al. 2020	X														
Surendra et al. 2020							X								
Shuai et al. 2020							X								
Rashid Waraich et al. 2014	X														
Simona Mancini 2017	X						X								
Sina et al. 2020			X												
Antti Lajunen fait en 2014	X														
Vincent Yu, et al. 2016	X														
Lu Zhen et al. 2019											X				
Eric Angel et al. proposent en 2014	X														
Albers Susanne propose en 2010	X														
Juho Andelmin et Enrico Bartolini 2017		X													

TABLE 2.2 – Récapitulatif de l'état de l'art du problème SMEPC (suite).

CHAPITRE 3

ETAT DE L'ART SUR LES MÉTHODES

Sommaire

3.1 Introduction	46
3.2 Généralités sur la modélisation	46
3.2.1 Généralités sur la programmation linéaire en nombres entiers	46
3.2.2 Généralités sur la technique de <i>Branch-And-Cut (B&C)</i>	47
3.3 Programmation dynamique	47
3.3.1 Cadre général	48
3.3.2 Exemples Simples	55
Le problème du sac à dos : Principe des FPTAS	55
Le <i>Split</i>	58
3.3.3 Le Cas Stochastique	58
Un Premier Niveau de Difficulté : Intégrer dans le Modèle le Comportement du Système Réel face aux aléas	59
Un deuxième Niveau de Difficulté : La Problématique des Corrélations	60
3.3.4 Le Cas Cyclique	61
3.4 Apprentissage automatique	63
3.4.1 La problématique générale de l'apprentissage	64
3.4.2 Les réseaux de neurones artificiels	64
Description générale et représentation graphique d'un réseau de neurones	65
Description du fonctionnement des réseaux de neurones	67
Difficultés liées à la mise en œuvre de réseau de neurones	67
3.4.3 L'API Keras de Tensorflow Keras	68
Les couches et les poids dans tf.keras	68
Relier deux couches de neurones de manière partielle	72
Fixer des poids dépendant des instances	75
3.5 Conclusion	76

3.1 Introduction

Dans ce chapitre, on propose un état de l'art sur deux outils particuliers que nous allons utilisés pour traiter le problème **SMEPC**. Ces deux outils sont la programmation dynamique et l'apprentissage automatique. On présente aussi quelques généralités sur les techniques de modélisation par programmation linéaire.

3.2 Généralités sur la modélisation

3.2.1 Généralités sur la programmation linéaire en nombres entiers

Un programme mathématique est un problème d'optimisation sous contraintes, c'est à dire qu'il consiste à minimiser (ou maximiser) une fonction sous contraintes. Un programme mathématique est un programme linéaire (PL) si la fonction-objectif et les conditions à respecter sont linéaires. On parle de programme linéaire en nombres entiers (PLNE) lorsque les valeurs des variables du programme linéaire doivent être des nombres entiers alors qu'un programme linéaire mixte (PLM) est obtenu lorsque certaines variables doivent être entières et d'autres pas. Modéliser un problème de tournées de véhicules par un PL ou PLNE revient donc à définir :

- les variables de décision, représentant les décisions qui doivent être prises, par exemple le choix du véhicule qui visite chaque client, l'ordre de visite des clients, etc.
- la fonction-objectif exprimée en fonction des variables, représentant le ou les objectifs à remplir, les termes à maximiser ou minimiser ;
- les contraintes exprimées en fonction des variables, représentant les conditions ou limites (capacité, logiques, temporelles, etc.) à prendre en compte lors de la construction de solutions réalisables. Les variables de décision et la fonction objectif à optimiser sont soumises aux contraintes . Il existe plusieurs types de contraintes :
 - les contraintes d'intégrités ou de signe ;
 - les contraintes d'inégalité par exemple $Ax \leq b$;
 - les contraintes d'égalité par exemple $Ax = b$.

On donne une définition plus formelle d'un PLNE à la définition (1). Relaxer un PLNE ou un PLM consiste en général à supprimer certaines de ses contraintes, ou alors les remplacer par des contraintes plus faibles. Le problème résultant est souvent plus simple à résoudre que l'original, mais si la relaxation est bien faite alors sa solution optimale peut être réalisable pour le problème original, et donc optimale pour ce dernier, ou alors ne pas être réalisable mais avoir un coût proche de l'optimum du problème original. On donne une définition plus formelle de la relaxation d'un PLNE à la définition (2).

Définition 1 [86] soient A une matrice de taille $n \times m$, b un vecteur colonne de taille m et c un vecteur ligne de taille n . Un programme linéaire en nombres entiers est le problème d'optimisation suivant :

$$(P) \left\{ \begin{array}{l} Ax \leq b \\ z = \text{Max}(cx) \text{ ou } \text{Min}(cx) \end{array} \right.$$

avec $x = (x_j \in \mathbb{N}, j = \{1, 2, \dots, n\})$, x est un vecteur inconnu de taille n et la fonction objectif z est une fonction linéaire à maximiser ou à minimiser. On dit qu'on a un programme linéaire en variables bivalentes si les contraintes $x_j \in \mathbb{N}$ sont remplacées par $x_j \in \{0, 1\}$.

Définition 2 [86] Quand on relâche les contraintes d'intégrité sur les variables ($x_j \in \mathbb{N}$) dans le programme linéaire en nombres entiers P de la définition 1, on obtient le programme linéaire suivant :

$$(P') \left\{ \begin{array}{l} Ax \leq b \\ z = \text{Max}(cx) \text{ ou } \text{Min}(cx) \end{array} \right.$$

avec $x_j \geq 0$, $j = \{1, 2, \dots, n\}$.

Il existe une relation entre P et P' : si par hasard, la solution optimale de P' est entière, c'est aussi une solution de P .

Généralement, les méthodes exactes permettent de trouver des solutions optimales en temps polynomial pour des problèmes de petite taille. Mais lorsque les problèmes sont de grande taille ces techniques ont du mal à trouver des solutions. Pour pouvoir traiter des problèmes de plus grande taille, on peut enrichir nos programmes avec certaines techniques comme par exemple le *Branch-And-Cut (B&C)*.

3.2.2 Généralités sur la technique de *Branch-And-Cut (B&C)*

La génération de coupes est une méthode utilisée quand le programme linéaire possède un très grand nombre de contraintes (par exemple exponentiel par rapport au nombre de variables). Soit P un programme linéaire (aussi appelé modèle dans la terminologie Cplex) et C l'ensemble des contraintes de P . L'idée est d'essayer de résoudre le modèle P sans être obligé d'y inclure explicitement toutes les contraintes. On forme alors le modèle initial P_0 avec un sous-ensemble $C_0 \subset C$ des contraintes, on le résout et on examine la solution. Si la solution est valide (i.e. toutes les contraintes de C sont respectées) alors on a la solution optimale. Sinon on ajoute au modèle les contraintes dans C violées par la solution. On recommence itérativement ce processus jusqu'à obtenir une solution qui respecte toutes les contraintes. On espère atteindre cette solution avant d'avoir intégré explicitement toutes les contraintes au modèle. Remarquons qu'on appelle coupes les contraintes ajoutées itérativement au modèle car elles coupent (i.e. rendent invalide) la solution courante.

3.3 Programmation dynamique

La *Programmation Dynamique* (PD) [15] [18] [53] [73] [77] constitue un outil, à la fois outil algorithmique et de modélisation, destiné à la gestion de *trajectoires*, c'est-à-dire de séquences de transitions d'un système qui obéit à une logique physique ou économique propre, mais sur lequel un acteur humain est supposé agir. Un point essentiel pour que la recherche de telles trajectoires soit justifiable est que leurs performances s'expriment comme une *somme* (application d'un opérateur associatif) de performances de leurs transitions. Dès lors, d'un point de vue algorithmique, les processus de programmation dynamique peuvent souvent être vus comme des processus de recherche de chemins dans un graphe sans circuit, ou le cas échéant de circuits quand les trajectoires sont cycliques, et font souvent dès lors référence au principe dit de Bellman. Dans le cas où le temps est continu, on parle souvent de *Contrôle Optimal*, et le principe de Bellman se décline alors sous forme d'équations différentielles.

Les applications sont multiples : Gestion de production et gestion de stock en contexte industriel, contrôle de trajectoire en aéronautique, gestion de portefeuille (contrôle optimal stochastique) en Finance de Marché, procédés dits du *Split* en Algorithmique de l'Optimisation de Tournées.

3.3.1 Cadre général

Afin de rendre plus lisible la présentation de ce cadre général, on va s'appuyer ici sur un exemple simple :

1. On considère une usine U qui produit chaque jour des quantités d'un produit unique P . Chaque soir, cette usine livre le contenu de sa commande au(x) client(s) : on travaille ici sur un ensemble de 30 jours, numérotés $i = 1, \dots, 30 = N$, et la quantité à livrer chaque soir est connue et de valeur L_i ;
2. L'usine est contrainte par des contraintes de capacité :
 - Capacité de stockage de la production d'un jour à l'autre : à un instant donné, l'usine de stockage ne peut accueillir plus de C^{Stock} unités de produit;
 - Capacité de production : sur une journée, U ne peut produire plus de C^{Prod} unités de produit. Ce rendement peut toutefois être augmenté jusqu'à un niveau $C^{Prod_Sup} > C^{Prod}$ dès lors que l'usine loue une certaine machine M .
3. L'usine a enfin à s'acquitter de coûts :
 - Coût de stockage : Si la quantité α de produit stockée est non nulle alors stocker α unités de produit pendant une nuit à un coût $Fix_Stock + \alpha \times Var_Stock$, sinon ce coût de stockage est nul.
 - Coût de production : Si la quantité β de produit réalisée au jour i est non nulle alors produire β unités de produit au jour i a un coût $Fix_Prod_i + \beta \times Var_Prod_i$, sinon ce coût de production est nul.
 - Coût de location/installation de machine, qui traduit le fait qu'utiliser la machine M a un coût :
 - Coût d'installation $Install_i$, si l'installation est réalisée durant la nuit qui conclut le jour i ; On considère que si M est utilisée deux (ou plus) jours consécutifs, elle n'a pas à être enlevée et donc elle n'est installée qu'une fois. Par contre, si elle cesse à un jour donné d'être utilisée (louée) alors elle doit être désinstallée et rendue (la désinstallation s'effectue le soir après utilisation, ce qui fait que si elle est à nouveau utilisée un peu plus tard, elle doit être à nouveau installée).
 - Coût de location par jour d'utilisation : Loc_i .
 - Compte tenu de tout cela, on veut donc produire et stocker de façon à satisfaire la demande (contraintes) tout en minimisant le coût (performance).

Cet exemple va nous permettre d'illustrer ce que sont les principaux composants d'un modèle (et aussi d'un algorithme) de *Programmation Dynamique*. Ceux-ci sont les suivants :

- Un Espace-Temps \mathcal{T} ;
- Un Espace d'Etats E ;
- Un Espace de Décisions D et de Transitions TR Associées;
- Une Stratégie d'Implémentation du Principe de Bellman;
- Les procédés de Filtrage.

Un Espace-Temps \mathcal{T} : Cet *espace-temps* peut être **continu**, assimilé en général à l'espace \mathbb{R} des nombres réels ou à un intervalle $[0, TMax]$. Il peut être aussi **discret**, muni d'un ordre partiel $<<$. Dans le cas où cet ordre est linéaire (Deux éléments sont toujours comparables), on pose souvent $\mathcal{T} = 0, \dots, N$. Il peut enfin être **cyclique**, ce qui signifie que le processus que l'on cherche à optimiser a vocation à être périodique. Dans ce dernier cas on considère implicitement que N coïncide avec 0 (ou $TMax$ avec 0).

Dans le cas de notre exemple, l'espace-temps sera bien entendu l'espace $\mathcal{T} = 0, 1, \dots, N, N + 1$ des jours, augmenté de deux jours fictifs 0 et $N + 1$ afin de pouvoir poser les bilans en début et fin de processus. Cet ensemble \mathcal{T} est discret et ordonné de façon linéaire. Dans nos algorithmes, on utilisera la notation i (au lieu de t) pour désigner un instant.

Un Espace d'Etats E : Un état $e \in E$, associé à un instant $t \in \mathcal{T}$, est une description de l'état du système considéré à cet instant t . Il est généralement fourni au travers d'un ensemble de variables prenant des valeurs numériques (indicateurs de stock, température, vitesse, position, etc) ou booléennes (propriétés), voire des valeurs plus complexes telles que des listes, représentatives par exemple d'historiques. De ce fait, une partie de la difficulté liée à la mise en œuvre de schémas DP tient au fait que ces états sont trop nombreux pour être énumérés, voire même infinis.

Parmi ces états se trouve généralement un **Etat Initial** e_0 , associé à l'instant initial $t = 0$, ainsi qu'une classe d'Etats Finaux, associé à l'instant N ou $TMax$, et caractérisés par une propriété particulière. Ceci est bien entendu remis en cause quand l'horizon est cyclique.

Il est important aussi de remarquer qu'à un instant donné t , tous les états de E ne sont pas forcément possibles. Il est donc important de parvenir à filtrer, c'est-à-dire à identifier à chaque instant t , l'ensemble $E(t)$ des états possibles et pertinents.

Dans le cas de notre exemple, il faut pour définir notre espace d'états, s'entendre sur le moment précis où cet état est constaté : un état est en fait une photo instantanée du système, et, le système étant évolutif, il est essentiel d'être précis sur l'instant auquel cette photo est prise. On considérera ici que l'état de notre usine est considéré au petit matin du jour i , avant qu'ait été lancée la production. Cet état est alors un vecteur e constitué de :

- e_S = Quantité de produit en Stock ;
- e_M = Etat de la machine : 1 si la machine est en place et 0 sinon.

On doit bien sûr avoir $e_S \leq C^{Stock}$.

L'état initial correspond au couple $i = 0, e_0 = (0, 0)$. Le fait d'introduire ainsi un jour fictif nous permet de traiter le cas de l'installation de la machine la veille du premier « vrai » jour $i = 1$. L'état final correspondra au jour $N + 1$ et à n'importe quel état (e_S, e_M) . En fait, on voit bien que si l'on n'envisage pas de continuation de l'activité, alors il est inutile d'avoir des stocks ou de garder la machine en fin de processus. Le seul état final utile est donc $e_f = (0, 0)$.

Un Espace de Décisions D et de Transitions TR Associées : A chaque instant t et pour tout état associé e , une décision d fournit les paramètres de la procédure qui va s'appliquer à e pour que le système se transforme en un état e' correspondant à un instant $t' > t$. Là encore, cette décision va être décrite au travers d'un vecteur de paramètres, numériques, booléens ou autres, et l'ensemble D pourra éventuellement être trop grand pour être énuméré. Également, tout l'espace D ne pourra pas forcément s'appliquer à l'instant t et l'état e , et il conviendra d'identifier le plus possible l'espace $D(t, e)$ des décisions à la fois réalisables et pertinentes.

Une décision d dans $D(t, e)$ étant choisie, la procédure de transformation de (t, e) vers (t', e') va alors s'appliquer, donnant lieu à ce qu'on nomme une **transition** notée $((t, e) \rightarrow_d (t', e'))$. L'ensemble

TR de ces transitions possibles n'est bien sûr jamais manipulé de façon explicite. Mais conceptuellement, il peut être assimilé à l'ensemble des arcs d'un graphe orienté dont les noeuds sont les couples (t, e) , et à ce titre facilite la compréhension des équations et la conception des algorithmes.

Une partie importante des difficultés sous-jacente à la Programmation Dynamique se trouve dans cette notion de transition. En effet :

- Si l'*Espace-Temps* \mathcal{T} est continu, alors une transition ne nous fait pas toujours passer d'un couple (t, e) à un couple (t', e') , mais souvent au contraire de (t, e) à $(t + dt, e')$, où dt ne désigne aucune valeur précise, mais seulement une variation infinitésimale de t .
- Une transition peut ne pas être *déterministe*, et de fait dans la pratique elle l'est rarement. Ce n'est donc pas un couple (t', e') qui résulte de l'application de la décision d à (t, e) , mais une distribution probabiliste sur un ensemble d'états résultats possibles. C'est le contexte de la *PD Stochastique* ou du *Contrôle Optimal Stochastique*, qui prévaut notamment dans l'univers de la gestion d'actifs financiers.

Dans le cas de notre exemple, il faut, comme pour la notion d'état, s'entendre sur l'instant où la décision est prise. En principe, elle doit l'être immédiatement après la prise de photo *Etat*, et de façon instantanée, ce qui ne correspond pas forcément à une réalité pratique. Elle est alors, pour un instant (jour) i donné, naturellement constituée d'un vecteur $d = (d_P, d_M)$ avec :

- d_P = quantité de production projetée pour le jour i ;
- $d_M = 0$ ou 1 : si $d_M = 1$ alors on change l'état de la machine et sinon on le maintient. Attention, cette décision ne concerne que la fin du jour i : l'état de la machine pendant le jour i est donc déjà fixé et égal à e_M .

On voit clairement que l'on doit avoir $d_P \leq C^{Prod} + (C^{Prod_Sup} - C^{Prod}) \times e_M$. Dans le cas $i = 0$, on doit par ailleurs avoir $d_P = 0$, puisque $i = 0$ est un jour fictif. La transition induite vient alors naturellement. Au matin du jour $i + 1$, on aura un nouvel état (e'_S, e'_M) , caractérisé par :

- $e'_S = e_S + d_P - L_i$;
- $e'_M = e_M + d_M$ (addition prise modulo 2).

On voit alors apparaître une pré-condition additionnelle quant à la faisabilité de la décision d :

- $C^{Stock} \geq e_S + d_P - L_i \geq 0$.

La figure (3.1) ci-dessous montre une représentation graphique (réseau) de notre système d'état et de transitions, correspondant à $i = 10$, $C^{Stock} = 3$, $C^{Prod} = 2$, $C^{Prod_Sup} = 4$, $L_{10} = 3$, $Fix_Prod_{10} = 3$, $Var_Prod_{10} = 2$, $Var_Stock = 1$, $Fix_Stock = 1$, $Install_{10} = 5$, $Loc_{10} = 2$.

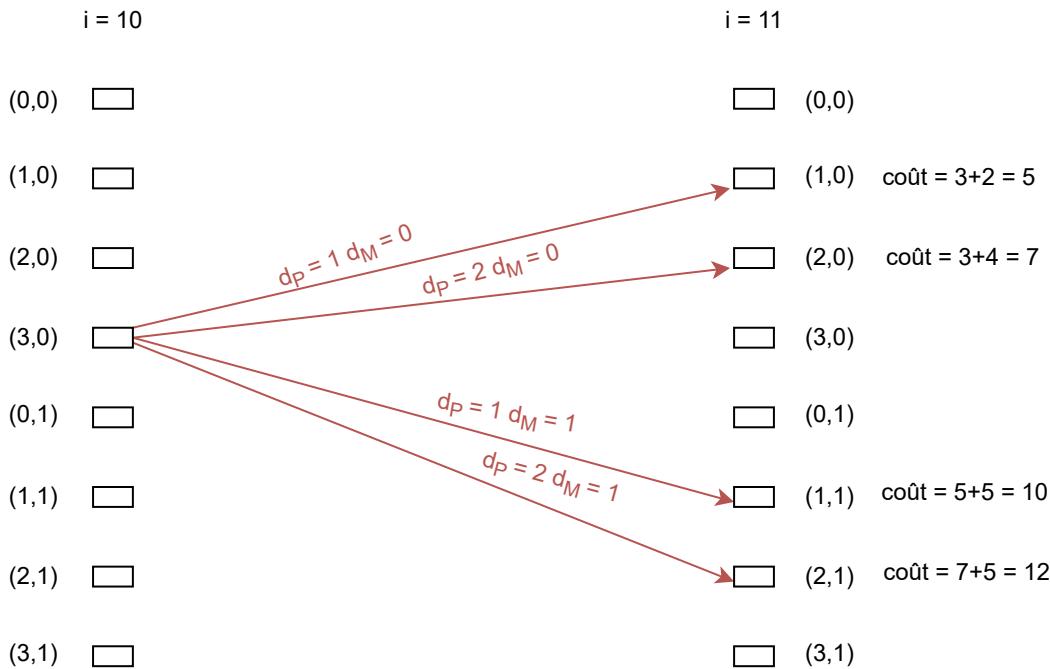


FIGURE 3.1 – Arcs associés aux Transitions dans le Réseau d’États associé à l’Exemple Production.

La figure (3.2) montre la structure générale du réseau des états, associé ici à \mathcal{T} et E .

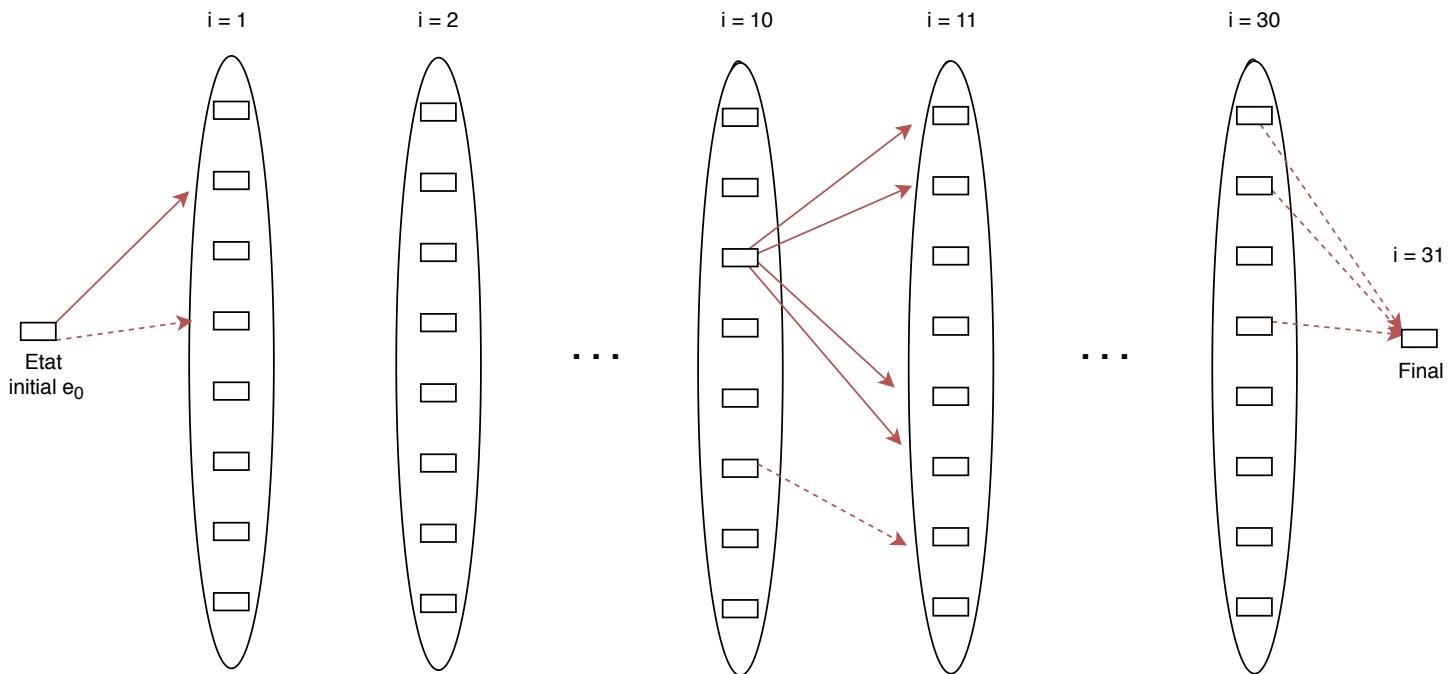


FIGURE 3.2 – Structure Générale du Réseau d’État «Production».

Mesure de Performance et Principe de Bellman : C'est un des points clés de la PD : une trajectoire au sens précédent est donc une suite de transitions $\Gamma = (t_0, e_0), \dots, (t_{Fin}, e_{Fin})$. A chaque transition $((t, e) \rightarrow_d (t', e'))$ de TR , il doit être possible d'associer une mesure de performance $Perf_Trans((t, e) \rightarrow_d (t', e'))$. **La mesure de performance de la trajectoire doit dès lors pouvoir s'exprimer** $Perf(\Gamma) = \otimes Perf_Trans((t_i, e_i) \rightarrow_{di} (t_{i+1}, e_{i+1}))$, où \otimes désigne un opérateur associatif (en général le $+$, mais pas forcément, voir par exemple les algèbres dites exotiques en automatique). Si on interprète l'ensemble des couples (t, e) comme l'ensemble des nœuds d'un graphe orienté, alors la notion de tra-

jectoire coïncide dans le cas déterministe et discret avec celle de chemin, et, quand l'opérateur \otimes est l'opérateur $+$, celle de performance avec celle de longueur de ce chemin.

C'est cette forme particulière de la mesure de performance d'une trajectoire qui fournit ce qu'on appelle le principe de Bellman. Dans le cas déterministe et discret, on va pouvoir noter $W(t, e)$, la performance optimale pour une trajectoire partielle aboutissant à l'instant t sur l'état e , et on aura alors : **(Equation Récursive de Bellman)**

$$W(t, e) = \text{Sup}_{\text{Transitions_possibles}((t'e') \rightarrow_d (t, e))} [W(t', e') \otimes \text{Perf_Trans}((t', e') \rightarrow_d (t, e))].$$

Les choses sont plus compliquées dans le cas stochastique ou encore dans le cas cyclique. On verra sur des exemples comment les choses peuvent alors se traiter.

Dans le cas de notre exemple, la performance devient un coût, qu'il faut minimiser. Dans la formule ci-dessus, il faut donc remplacer Sup par Inf, l'opérateur \otimes devenant de façon classique l'opérateur $+$. Le coût de la transition $((e_S, e_M) \rightarrow_d (e'_S, e'_M))$, réalisée entre i et $i + 1$ va être :

Cout_Transition = Cout_Stock + Cout_Prod + Cout_Machine avec :

- Si $d_P = 0$ alors $Cout_Prod = 0$ sinon $Cout_Prod = Fix_Prod_i + d_P \times Var_Prod_i$;
- Si $(e_S + d_P - L_i) = 0$ alors $Cout_Stock = 0$ sinon $Cout_Stock = (e_S + d_P - L_i) \times Var_Stock + Fix_Stock$;
- $Cout_Machine = (1 - e_M) \times d_M \times Install_i + e_M \times Loc_i$.

Stratégie d'Implémentation du Principe de Bellman : Concevoir un algorithme de Programmation Dynamique consiste donc à implémenter le *Principe Récursif de Bellman*, tel que décrit ci-dessus. Suivant le cas, on pourra se projeter en avant (*Forward Driven*) à partir de l'état initial ou au contraire en arrière (*Backward Driven*) depuis les états finaux possibles, en encore adopter une stratégie mixte, dite *Forward/Backward*, de façon à réduire le nombre des états créés. Il est cependant important de noter que des considérations autres qu'algorithmatiques peuvent présider au choix d'une telle stratégie. Dans un certain nombre de cas, et notamment dans le cas des applications temps réel, l'algorithme PD n'est appliqué qu'en pré-process, de façon à obtenir, pour chaque couple (t, e) , la décision optimale $d(t, e)$ associée. Cette décision est alors stockée dans une forme de table, et c'est la procédure de récupération en temps réel de cette décision qui est alors exécutée à chaque instant au fur et à mesure que la trajectoire se déroule. Outre la réactivité que cette façon de procéder apporte, celle-ci permet aussi de gérer les éventuelles perturbations, c'est-à-dire le fait que les transitions puissent induire des marges d'erreurs. Mais elle implique une implémentation *Backward* du principe de Bellman, puisque c'est celle-ci qui va nous permettre de récupérer les décisions souhaitées :

Equation Récursive de Bellman, en Forme Backward : Pour tout couple t, e , on note $V(t, e)$ la valeur de performance optimale d'une séquence de transitions permettant de connecter l'état e à l'instant t à un état final à l'instant final N . On obtient alors :

- $V(t, e) = \text{Sup}_{\text{Transitions_possibles}((t, e) \rightarrow_d (t', e'))} [V(t', e') \otimes \text{Perf_Trans}((t, e) \rightarrow_d (t', e'))]$;
- Décision associée $d(t, e) = \text{Arg Sup}$.

Dans le cas de notre exemple, une implémentation Backward du principe de Bellman se fera en construisant, pour $i = N + 1, N, \dots, 1, 0$, une liste $E(i)$ des états associés à l'instant i , dont chaque élément est constitué de :

- Un état e (état à l'instant i) ;
- Une valeur associée V , nulle dans le cas où $i = N + 1$;
- Une décision associée d , prise à l'instant i , dans l'état e et indéfinie dans le cas où $i = N + 1$;

- Un pointeur pt sur l'élément de la liste $E(i + 1)$, qui correspond au résultat de la transition induite par d depuis (i, e) .

Le processus fonctionnera alors comme décrit à l'algorithme (1).

Algorithme 1 Backward_DPS

```

1: Initialiser  $E(N + 1) \leftarrow \{(e_f, 0, Indfini, Nil)\}$  ;
2: pour  $i = N$  à 0 faire
3:   pour  $(e', V', d', pt')$  dans  $E(i + 1)$  faire
4:     Générer les états  $e$  et les décisions  $d$  tels que la transition  $(e \rightarrow_d e')$  est réalisable ;
5:     pour tout couple  $(e, d)$  ainsi généré faire
6:       Si  $e$  n'apparaît pas dans  $E(i)$  alors
7:         insérer  $(e, V, d, pt)$  dans  $E(i)$  avec  $V = V' + Cout\_Transition((i, e) \rightarrow_d (i + 1, e'))$  et  $pt =$ 
          Pointeur sur  $(e', V', d', pt')$  dans  $E(i + 1)$  ;
8:       sinon
9:         Soit  $(e, V^*, d^*, pt^*)$  l'élément mettant en jeu  $e$  dans  $E(i)$  ;
10:        Si  $V = V' + Cout\_Transition((i, e) \rightarrow_d (i + 1, e')) < V^*$  alors
11:          Remplacer  $V^*$  par  $V$ ,  $d^*$  par  $d$  et  $pt^*$  par  $pt$  = Pointeur sur  $(e', V', d', pt')$  dans  $E(i + 1)$  ;
12:        Fin si
13:        Fin si
14:      fin pour
15:    fin pour
16:  fin pour

```

Filtrage : Comme évoqué plus haut, une des difficultés majeures à traiter quand on cherche à implémenter un algorithme **PD** est liée à l'explosion potentielle du nombre des états. Se pose alors la question du **Filtrage**, c'est-à-dire des procédés susceptibles d'anticiper les états les plus pertinents. On distinguera quatre grandes classes de filtrage :

- *Les filtrages par dominance* : On définit, soit de façon mathématiquement fondée, soit de façon heuristique, un procédé de *comparaison qualitative* des états, qui induit en général une relation d'ordre sur E , et ne conserve parmi les états générés pour une valeur t donnée, que les états maximaux ou presque maximaux pour ce procédé.
- *Les filtrages par équivalence ou rounding* : On définit un procédé qui consiste à considérer comme équivalents entre eux deux états e_1 et e_2 « Proches ». On ne conserve alors, parmi les états générés pour une valeur t donnée, qu'au plus un état par classe d'équivalence induite par ce procédé.
- *Les filtrages par anticipation logique* : On se dote de procédés de déduction (similaire à ceux utilisés pour la *Propagation de Contraintes*), permettant d'anticiper que, à un instant donné t , il ne sera pas possible de prolonger (vers l'état final si on est en *Forward* et vers l'état initial si on est en *Backward*) un certain état e en une trajectoire Γ satisfaisant toutes les contraintes du problème.
- *Les filtrages par estimation optimiste* : On se dote d'un procédé fournissant, pour tout couple (t, e) , une *estimation optimiste* $Val(t, e)$, (en général une borne supérieure) de la meilleure performance possible pour une trajectoire partielle $\Gamma(e, t)$, démarrant en (e, t) et se poursuivant jusqu'à un état final possible (si on est en *Forward* et terminant en (e, t) si on est en *Backward*). Dès lors, pour peu que l'on ait été capable de pré-calculer de façon heuristique une trajectoire réalisable de valeur $WCour$, on peut appliquer le filtrage suivant :

- si (t, e) est tel $W(t, e) \otimes Val(t, e) \leq VCour$, $W(t, e)$ désignant la valeur associée à (t, e) au sens de la formule de Bellman déclinée *Forward*, alors tuer e en tant qu'état possiblement associé à t .

Remarquons au passage que le fait de disposer du procédé d'estimation optimiste Val permet de transformer de façon naturelle le schéma **PD Forward** en un algorithme glouton, en implémentant le processus suivant :

- $(t, e) \leftarrow (t_0, e_0)$;
- Tant que (t, e) ne correspond pas à un état final faire :
 1. Calculer une décision d telle que $W(t, e) \otimes Perf_Trans((t, e) \rightarrow_d (t', e')) \otimes Val(t', e')$ soit maximal;
 2. $(t, e) \leftarrow Etat(t', e')$ résultat de l'application de d à (t, e) .

Dans le cas de notre exemple, on voit que la structure des états est très simple, essentiellement parce qu'on considère un seul produit *output* et que l'on ne s'intéresse pas aux produits *input*, et que l'on gère une production agencée de façon séquentielle dans le temps, chaque opération de production se déroulant sur une seule période i . Malgré cela on peut être confronté à un nombre important d'états pour peu que les unités utilisées pour mesurer production et stockage soient fines, et donc que l'espace des valeurs pour les variables e_P et d_P soient grands. Les notions évoquées ci-dessus pourront alors s'exprimer comme suit (on suppose que l'on se place dans l'hypothèse d'un parcours *Forward*) :

- Filtrage par Dominance* : pour un instant i considéré, on ne voit pas comment de façon mathématique dire qu'un état $e = (e_S, e_M)$ accompagné d'une valeur W , est meilleur qu'un état $e^* = (e_S^*, e_M^*)$ accompagné d'une valeur W^* . En effet, le fait de disposer de la machine ou d'un stock élevé n'est pas forcément un avantage, puisqu'il faudra payer pour les coûts de location et stockage induits. Toutefois, on peut considérer, de façon empirique, que si la demande à fournir entre i et N est élevée et si les coûts de stockage sont faibles en comparaison des coûts de production, alors le fait d'avoir $e_M^* \geq e_M$, $W^* \leq W$ et $(e_S^* - e_S)$ au moins égal à un certain seuil rend un élément (e^*, W^*, d^*, pt^*) de $E(i)$ meilleur que l'élément (e, W, d, pt) . On retire (tue) alors ce dernier de $E(i)$.
- Filtrage par Rounding* : Il suffit ici de changer d'unité de mesure des produits (par exemple, de compter en tonnes au lieu de compter en kilos. On se fixera donc un nombre de digits K et considérera que si (e^*, W^*, d^*, pt^*) et (e, W, d, pt) dans $E(i)$ sont tels que $e_M = e_M^*$, $e_S = e_S^*$ et $W = W^*$ au K plus grands digits, alors ils sont équivalents. On en supprimera donc un des deux.
- Filtrage par Anticipation Logique* : La puissance d'un tel filtrage dépend, comme en programmation par contrainte, de l'habileté du programmeur à se projeter en avant. Il existe donc autant de procédés que de programmeurs habiles. De façon extrêmement rustique, on pourra remarquer ici que si à un instant i donné, la somme $\sum_{j \geq i} L_j$ des livraisons à effectuer est strictement plus grande que la capacité de production restante augmentée du stock courant, soit $(N - i + 1) \times C^{Prod_Sup} + e_S$, alors il ne sera pas possible de prolonger l'état e à l'instant i en une solution de notre problème. On pourra donc tuer le composant (e, W, d, pt) associé dans $E(i)$.
- Filtrage par Estimation Optimiste* : Il en est de même que pour les filtrages par anticipation logique. Leur puissance dépend de la capacité du programmeur à concevoir un procédé d'estimation $Val(t, e)$ qui fournit une bonne approximation du coût à consentir afin de prolonger

un état e à un instant t en une trajectoire complète. Ici, on pourra toujours remarquer (là encore de façon très rustique), que si on est dans un état (e_S, e_M) à un instant i , et si $L = \sum_{j \geq i} L_j$ est la quantité demeurant à livrer, alors il faudra produire au moins $(L - e_S)$ quantité de produit, sur au moins $\lceil (L - e_S)/C^{Prod_Sup} \rceil$ journées. Le coût induit sera donc au moins égal à $Val(i, e) = (Inf_{j \geq i} Fix_Prod_j) \times \lceil (L - e_S)/C^{Prod_Sup} \rceil + (Inf_{j \geq i} Var_Prod_j) \times (L - e_S)$. Il faudra toutefois chercher quelque chose de plus sophistiqué si on veut vraiment obtenir un effet de filtrage conséquent.

3.3.2 Exemples Simples

On va à présent illustrer ces notions générales au travers d'un certain nombre d'exemples, simples dans un premier temps (Section 3.3.2), puis plus complexes (Section 3.3.3, 3.3.4) et traitant notamment du cas stochastique (Section 3.3.3), et du cas cyclique (Section 3.3.4).

Le problème du sac à dos : Principe des FPTAS

Le problème du sac à dos consiste en la recherche d'un vecteur $x = (x_1, \dots, x_N)$ à valeur en $\{0, 1\}$ et tel que :

- $\sum_i A_i \times x_i \leq B$;
- il faut maximiser $\sum_i C_i \times x_i$.

Les coefficients entiers $A_i, C_i, i = 1, \dots, N$, et B sont donnés en entrées.

On notera qu'aucune notion évidente de temps n'est présente dans cet exemple, qui ne concerne pas un système dynamique. Pour autant, un espace-temps artificiel s'impose, qui est l'espace $\mathcal{T} = \{1, \dots, N, N + 1\}$.

L'espace d'état associé est alors l'espace E des valeurs entières e entre 0 et B . Sa signification est la suivante : à l'instant $i = 0, \dots, N + 1$, on a décidé des valeurs $x_j, j < i$, et la somme $\sum_{j < i} A_j \times x_j$ vaut e . On doit alors décider de la valeur x_i .

L'état initial correspond à $i = 1$ et $e = 0$. Les états finaux correspondent à $i = N + 1$ et $e \leq B$. Les noeuds du réseau sans circuit associés à ces couples Temps/Etats sont formés de tous les couples (i, e) ainsi obtenus. La figure 3.3 explicite le passage des noeuds du réseau des états du problème du sac à dos associés à $i = 3$ à ceux associés à $i = 4$, quand $B = 100$, $A_3 = 5$, $C_3 = 4$.

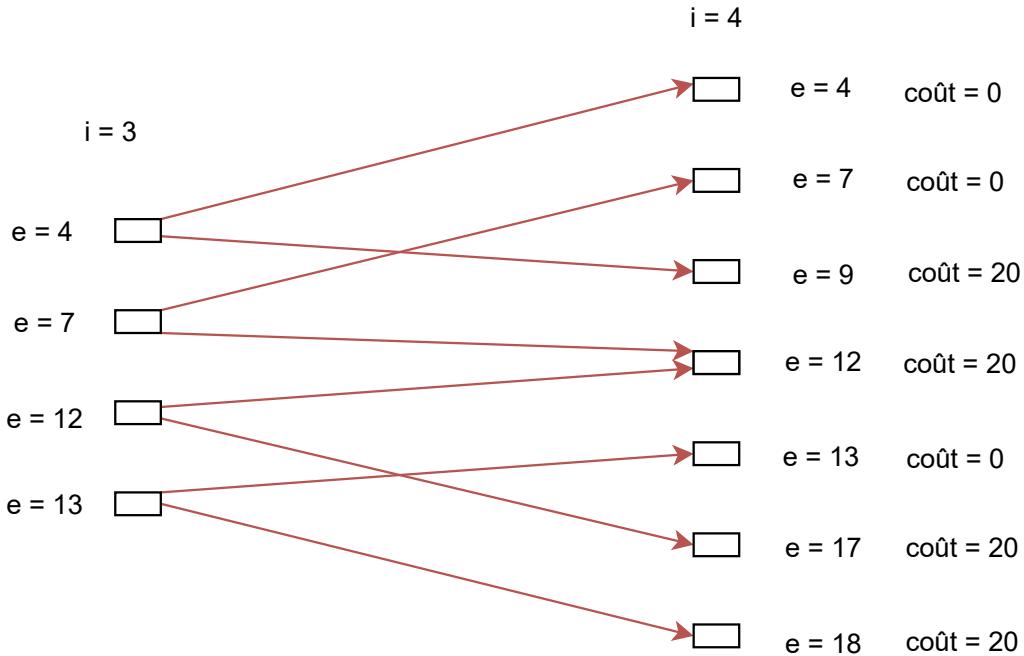


FIGURE 3.3 – Les nœuds du Réseau des Etats du problème du sac à dos.

Une décision prise à l'instant i depuis l'état e porte sur la valeur x_i . Elle est réalisable si $e + A_i \times x_i \leq B$. La transition (arc du réseau) induite fait passer de i en $i + 1$ et de e à $e + A_i \times x_i$. Son profit est $C_i \times x_i$. On retient, parmi les états finaux possibles e , celui qui est associé à la plus grande valeur de profit cumulé w , qui constitue donc la valeur optimale V^{Opt} de notre problème du sac à dos.

L'algorithme de **PD**, que l'on notera **PD-Knap**, qui en découle n'est pas polynomial (on sait que le problème du sac à dos est NP-Difficile) : en effet le nombre d'états à priori possibles est B , et est donc en $2^{k(B)}$, où $k(B)$ est la taille de codage de l'entier B . Il est toutefois presque polynomial : si l'on limite les valeurs de B de telle sorte qu'elles n'excèdent une valeur $Q(N)$, où Q est un polynôme, alors on voit que cet algorithme devient polynomial. Cela suggère à rendre notre algorithme presque polynomial, en arrondissant les valeurs prises par les états e , de telle sorte que leur nombre demeure polynomial en fonction de N , quitte à consentir une marge d'erreur sur le résultat final. C'est le sens de ce qu'on nomme les **PTAS** (**Polynomial Time Approximation Scheme**).

Construire un **PTAS** (**Polynomial Time Approximation Scheme**) consiste à construire un algorithme paramétré, qui une fois la valeur du paramètre fixée, devient polynomial au sens du temps, modulo un certain niveau d'approximation. On va jouer ici sur les tailles de codage et transformer l'algorithme **PD-Knap** ci-dessus comme suit :

- On se fixe un nombre K , qui est un nombre de bits.
- Pour chaque nombre entier p , dont l'écriture en base 2 est $\sum_{j=0, \dots, k(p)} u_j \times 2^j$, on pose $Round(K, p) = \sum_{j=k(p)-K+1, \dots, k(p)} u_j \times 2^j$, c'est-à-dire que l'on ne garde de p que ses K bits les plus élevés (on dit aussi que l'on arrondit aux K plus grands bits). On dit que 2 couples (a, w) et (a', w') de nombres entiers sont alors équivalents modulo les K plus grands bits si $Round(K, a) = Round(K, a')$ et $Round(K, w) = Round(K, w')$.
- On adapte alors le schéma **PD** précédent en gardant l'espace temps $\mathcal{T} = 1, \dots, N + 1$ et en prenant comme espace d'état E l'ensemble des couples $e = (a \leq B, w)$: à chaque instant i , on a décidé des valeurs x_j , $j < i$, la somme $\sum_{j < i} A_j \times x_j$ vaut a et la somme $\sum_{j < i} C_j \times x_j$ vaut w .

On doit alors décider de la valeur x_i . La décision x_i étant choisie, on passe alors (transition) à l'instant $i + 1$ dans l'état $e' = (\sum_{j \leq i} A_j \times x_j, \sum_{j \leq i} C_j \times x_j)$. On exige que $\sum_{j \leq i} A_j \times x_j \leq B$.

- Durant le passage $i \rightarrow i + 1$, on applique le filtrage suivant :
 - On implémente **PD** selon la stratégie *Forward* ;
 - On fait en sorte, que, dans l'espace $E(i + 1)$ des états associés à l'instant $i + 1$ à l'issue du passage $i \rightarrow i + 1$, **il n'y ait pas 2 états e' et e'' qui soient équivalents modulo les K plus grands bits**. Si, à partir de $E(i)$, on a fait apparaître e' et e'' équivalents modulo les K plus grands bits, alors **on élimine celui qui a le plus grand composant** $a = \sum_{j \leq i} A_j \times x_j$.
- On retient, parmi les états (a, w) obtenus avec $i = N + 1$, celui qui correspond à la plus grande valeur w , qui fournit donc la valeur de l'algorithme.

On obtient ce faisant, un algorithme paramétré par le nombre K . On le note **PD-Knap(K)**. Si K est fixé, cet algorithme devient polynomial au sens temps, puisque le nombre d'éléments dans chaque ensemble $E(i)$ ne peut excéder $2^{2K} \times (k(B) - K + 1) \times (k(C) - K + 1)$, où $C = \sum_i C_i$. En même temps, on peut rendre cet algorithme aussi proche de l'optimalité que l'on veut en augmentant le nombre K .

La figure (3.4) explique comment on regroupe ainsi les états par classe d'équivalence modulo les $K = 3$ plus grands bits :

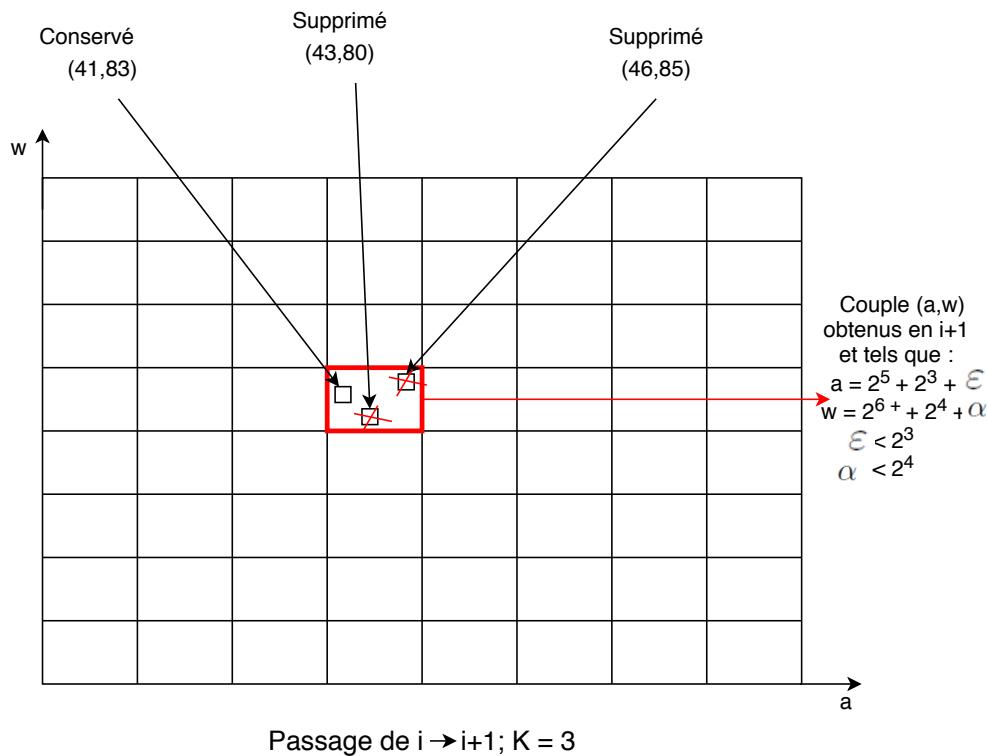


FIGURE 3.4 – Regroupement des états (a, w) par classe d'équivalence modulo les 3 plus grands bits.

Théorème 1 PTAS du Knapsack

Pour tout $\varepsilon > 0$, on peut concevoir un algorithme P^ε tel que pour toute instance I de valeur optimale V^{Opt} de notre problème du Knapsack on ait :

- P^ε calcule une solution réalisable x de valeur V de I en temps polynomial ;
- Le quotient « Erreur relative commise par P^ε » $\frac{V^{Opt} - V}{V^{Opt}} \leq \varepsilon$.

Le *Split*

On parle de procédé *Split* quand on cherche à partitionner une liste L en sous-listes L_1, \dots, L_Q , constituant chacune un segment de L (constituée d'éléments consécutifs dans L), de façon à minimiser un coût défini par les jointures entre les différentes listes. Il est utilisé en algorithmique des problèmes dits de Tournées de Véhicules, quand on cherche à distribuer une liste de clients L (notion de tour géant) donnés dans un certain ordre, entre des véhicules qui assureront le service d'un sous-ensemble de clients consécutifs au sens de L .

Différentes variantes du *Split* existent, plus ou moins complexes selon que l'on envisage ou non différentes catégories et suivant la façon dont on définit les coûts. On va prendre ici le cas le plus simple, sachant que, dans tous les cas, l'algorithme du *Split* reste sensiblement le même.

Considérons donc notre liste L , que l'on peut imaginer comme un tour $\{0 = Depot, 1, 2, \dots, N, N + 1 = 0 = Depot\}$ réalisé par un véhicule unique, partant d'un Dépôt nommé *Depot* et visitant dans cet ordre des clients $i = 1, \dots, N$, avant de revenir au Dépôt. Cet ensemble $X = \{0, 1, \dots, N\}$ est muni d'une distance : $D_{i,j}$ désigne la distance de i à j . A chaque client i correspond une charge C_i . Un nombre K étant donné, on cherche alors à distribuer ces clients entre K véhicules identiques, de capacité CAP , de façon à ce que les contraintes de capacités ne soient pas violées et que :

- Problème **SplitMax** : le Max des temps de parcours des véhicules soit le plus petit possible ;
- Problème **SplitSum** : la somme des temps de parcours des véhicules soit la plus petite possible.

Dans les deux cas, notre espace-temps va être l'espace ordonné $0, 1, \dots, N, N + 1$ et l'espace E des états va être composé des valeurs $e = 0, 1, \dots, K$, avec, pour un tel état e associé à un instant i , la signification suivante :

- On a traité tous les clients j tels que $1 \leq j \leq i$ (si $i = 0$ il n'y a pas de tels clients), et cela en utilisant e véhicules. Si $i \leq N$ on doit avoir $e \leq K - 1$.
- L'état initial e_0 est bien sûr $e_0 = 0$; L'état final est tout e associé à $N + 1$ et tel que $e \leq K$.

Une décision est alors définie par un indice j tel que : $i + 1 \leq j \leq N + 1$, tel que :

- $\sum_{i+1 \leq q \leq j} C_q \leq CAP$; (on pose $C_0 = C_{N+1} = 0$)
- Si $e = K - 1$ alors $j = N + 1$;

La transition induite va de (i, e) à $(j, e + 1)$. Son coût est $D_{Depot,i+1} + \sum_{i+1 \leq q \leq j-1} D_{q,q+1} + D_{j,Depot}$.

Le distinguo entre **SplitMax** et **SplitSum**, se trouve au niveau de l'opérateur associatif \otimes qui combine les coûts des différentes transitions aux fins d'obtenir un coût cumulé. Dans le premier cas, \otimes est l'opérateur Max, et dans le deuxième, il s'agit classiquement de l'opérateur $+$. Dans ce dernier cas aussi, on voit que l'on peut simplifier la spécification du coût des transitions en ne tenant compte que des détours induits par le changement de véhicule, c'est-à-dire des termes $D_{Depot,i+1} + D_{j,Depot}$.

3.3.3 Le Cas Stochastique

Il est au cœur de nombreuses situations réelles, notamment celles liées à la gestion d'actifs financiers. Nous n'allons faire ici que l'effleurer, de façon à en faire percevoir la complexité.

Reprendons l'exemple de la Section (3.3.1) et lié à la gestion de production. Dans la plupart des situations concrètes, il existe des incertitudes quant à la fraction d'objets produits qui ne pourront être livrés du fait de défauts, et, surtout, quant à la demande. Même quand les demandes sont associées à des commandes, celles-ci peuvent être annulées ou reportées, ou au contraire, des demandes non prévues peuvent apparaître qui nécessitent une certaine réactivité de la part de l'entreprise. Dans notre

cas, on peut imaginer que, pour chaque jour $i = 1, \dots, N$, la demande L_i soit incertaine, et qu'une analyse statistique des historiques permette de quantifier cette incertitude sous forme probabiliste : pour chaque i , L_i devient une distribution de probabilité à l'intérieur d'un espace D_i de valeurs possibles de demandes pour le soir i . Par exemple, pour le jour $i = 10$, on pourra supposer que $D_{10} = \{4, 5, 6\}$ et que L_{10} est la distribution de probabilité $\{1/4, 1/2, 1/4\}$. On supposera aussi $Fix_Prod_{10} = 10$, $Var_Prod_{10} = 4$, $Fix_Stock = 1$, $Var_Stock = 1$.

Un Premier Niveau de Difficulté : Intégrer dans le Modèle le Comportement du Système Réel face aux aléas

Conceptuellement, l'introduction du non déterminisme signifie qu'à une décision donnée correspond, non pas une transition unique, mais plusieurs, subordonnées à l'évènement « demande en i » $d \in D_i$ (voir Figure 3.5). Si, par exemple, on est dans un état $(e_S = 3, e_M = 0)$, et que l'on prend une décision $(d_P = 2, d_M = 0)$, alors cette décision est susceptible d'avoir trois résultats possibles à l'instant $i + 1$:

- Avec probabilité $1/4$: le stock résultant est 1 : le coût de production induit est 18, le coût de stockage induit est 2 ;
- Avec probabilité $1/2$: le stock résultant est 0 : le coût de production induit est 18, le coût de stockage induit est 0 ;
- Avec probabilité $1/4$: le stock résultant est 0 : le coût de production induit est 18, le coût de stockage induit est 0, mais, on a été incapable de satisfaire une demande.

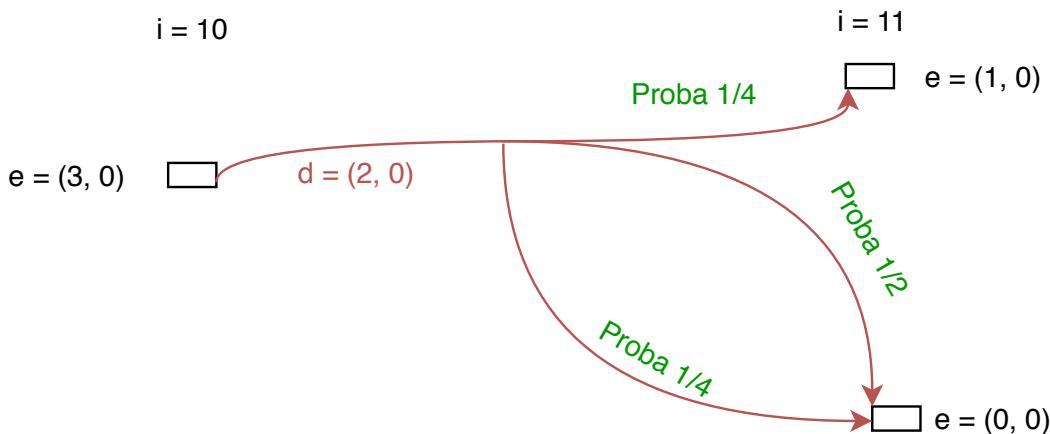


FIGURE 3.5 – Un arc-transition non déterministe.

Supposons maintenant que l'on a une capacité de stockage de 4, et que par souci d'anticipation des coûts favorables, on ait décidé de produire $d_P = 6$. Alors on n'a plus de problème de demande non satisfaite, mais on a **un excès de stock** dans le cas (Avec probabilité $1/4$) ou la demande en i est seulement de 4. Doit-on en déduire que la décision est forcément non réalisable ?

Cette première analyse fait apparaître un premier niveau de difficulté : afin d'intégrer les incertitudes, on est amené à élargir le modèle, de façon à intégrer le fait que certaines contraintes du modèle déterministe (satisfaction des demandes ou des capacités) ne peuvent plus forcément être considérées comme des contraintes, sauf à rendre le modèle trop rigide, et doivent être aménagées de façon à coller avec la réalité du système :

- Demandes non satisfaites : on peut les ignorer, les décaler, ou au contraire les intégrer en tant que pénalités si on considère qu'elles induisent une perte d'image (difficile à quantifier) ;
- Non respect des capacités de stockage : pratiquement, il faut regarder si ces contraintes sont « dures » (peut-on « faire de la place »?), si il est possible de louer de la place de stockage additionnelle auprès d'un partenaire (\Rightarrow coût additionnel), ou s'il faut détruire la production excédentaire (coût de destruction?).

La réponse à ces questions ne peut être fournie à partir du seul modèle déterministe, et exige un retour sur le terrain.

Un deuxième Niveau de Difficulté : La Problématique des Corrélations

Supposons à présent que l'on ait réussi à contourner les obstacles soulevés à la section 3.3.3, et donc que l'on ait fait en sorte que chaque décision admissible d prise depuis un état e à un instant i induise une transition TR , pour laquelle chaque évènement ω induira à son tour un état e' résultant $TR(i, e, \omega)$, selon un probabilité $P_\omega(e')$ et un coût $Cout_TR(i, e, e')$. Si l'on se réfère à l'exemple de gestion de production, on peut imaginer avoir pénalisé, par un coût additionnel, toute demande non satisfaite ainsi que tout dépassement de capacité (ce qui revient à supprimer les capacités pour les remplacer complètement par des coûts).

A ce moment là, on est en mesure d'implémenter un schéma **PD**, en se focalisant sur l'espérance (au sens probabiliste) du coût d'une stratégie de gestion :

- On note, pour tout instant i , et tout état e associé, $WP(i, e)$, l'espérance conditionnelle de Coût pour une trajectoire optimale débutant à l'instant i et se finissant à l'instant N final, et cela à partir de l'état e .
- On remarque alors que la seule stratégie de recherche possible pour l'implémentation d'une telle définition est la stratégie *Backward*, qui seule permet de tirer partie de la façon dont les transitions sont probabilisées (depuis l'état de départ vers les états résultats). Pour tout instant i , tout état e associé et toute décision d , on peut écrire que, si l'on est en mesure d'appliquer la stratégie optimale associée à WP à partir de tout instant ultérieur à i , alors le coût induit par la décision d sera : $Cout_Trajectoire(d, i, e) = \sum_{e'} P_\omega(e') \times (Cout_Transition(((i, e) \rightarrow_{d, \omega} (i', e')) + WP(i', e')))$.

Notre principe de Bellman s'adapte alors sans difficulté, sous la forme :

- $WP(i, e) = \inf_d Cout_Trajectoire(d, i, e) = \inf_d \sum_{e'} P_\omega(e') \times (Cout_Transition(((i, e) \rightarrow_{d, \omega} (i', e')) + WP(i', e')))$. Où d représente toutes les décisions possibles.
- Décision associée $d(t, e) = \arg \inf.$

Sur notre exemple : Supposons que l'on soit en $i = 10$, dans un état $e = (e_S = 1, e_M = 0)$, avec $D_i = \{3, 5, 7\}$, $L_i = \{1/4, 1/2, 1/2\}$ la distribution de probabilité associée. On suppose $Fix_Prod_{10} = 10$, $Var_Prod_{10} = 4$, $Fix_Stock = 1$, $Var_Stock = 1$, ainsi qu'une capacité de stockage de 2. On a enfin convenu de pénaliser de 2 unité-coût/unité-stockée tout dépassement de capacité et de 3 unités-coûts/unité-demandée tout manquement à la demande. Si l'on prend alors la décision ($d_P = 4, d_M = 0$), alors cette décision est susceptible d'avoir 3 résultats possibles à l'instant $i + 1$:

- Avec probabilité 1/4 : le stock résultant est 3 : le coût de production induit est 26, le coût de stockage (pénalisé) induit est 5 donc le Total est 31 ; Etat résultant $e_1 = (3, 0)$;

- Avec probabilité 1/2 : le stock résultant est 1 : le coût de production induit est 26, le coût de stockage induit est 2 donc le Total est 28 ; Etat résultant $e_2 = (1, 0)$;
- Avec probabilité 1/4 : le stock résultant est 0 : le coût de production induit est 26, le coût de stockage induit est 0 ; la pénalité liée à la demande est 3 donc le Total est 29 ; Etat résultant $e_3 = (0, 0)$.

Si l'on suppose que l'on a su calculer, pour $i = 11$, les valeurs $WP(11, e_1) = 108$, $WP(11, e_2) = 111$, $WP(11, e_3) = 106$, alors on voit que $Cout_Trajectoire(d, i, e) = (31 + 108)/4 + (28 + 111)/2 + (29 + 106)/4 = 138$.

Mais une difficulté apparaît alors : La formule

$Cout_Trajectoire(d, i, e) = \sum'_e P_\omega(e') \times (Cout_Transition(((i, e) \rightarrow_{d, \omega} (i', e')) + WP(i', e'))$ qui nous permet ce calcul repose sur l'hypothèse d'*indépendance*. Les différents évènements susceptibles d'intervenir durant le processus doivent être *indépendants* (ou Markovien ou *sans mémoire*), ce qui permet d'écrire que si P_A et P_B sont les probabilités que 2 d'entre eux, A et B, interviennent, alors la probabilité que les 2 interviennent au cours du même processus est égale au produit $P_A \times P_B$. Cette hypothèse, présente aussi dans les modèles de gestion d'actifs financiers reposant sur le *Mouvement Brownien*, facilite considérablement les calculs. Mais elle correspond rarement à la réalité, dans la mesure où elle nie la relation de causalité. Dans le cas par exemple de notre système de production, le fait que la demande à l'instant 10 soit 3 plutôt que 5 peut avoir plusieurs causes : l'une d'entre elles est que la demande manquante est l'objet d'un délai, et qu'on va la retrouver à l'instant $i+1$ sous forme d'une augmentation de la demande prévue en 11 ; une autre, qui va en sens opposé, est que la demande a fléchi parce que le produit est apprécié de façon négative, ou parce qu'un produit concurrent moins cher (ou meilleur) est en train d'émerger, ou encore à cause de la conjoncture globale. Saisir ces causalités, et donc les *corrélations* qui peuvent exister entre les distributions de probabilités L_i , est en soi difficile. Les intégrer à des calculs l'est encore plus. Une tendance actuellement est de remplacer des modèles basés sur des équations de type Bellman mettant en jeu ces hypothèses Markoviennes par des mécanismes d'apprentissage statistique exploitant des historiques.

3.3.4 Le Cas Cyclique

On ne consacrera que peu de place ici à ce dernier cas, malgré son importance pratique. Dans beaucoup de cas, notamment celui des systèmes de gestion de stock et de production, ou bien celui de certains services hospitaliers, on ne peut identifier clairement un début ou une fin. Le système fonctionne selon une certaine forme de périodicité (jour, semaine, mois, année, etc), et il faut organiser au mieux cette périodicité.

Afin de fixer les idées, on peut reprendre l'exemple initial du système de production, en considérant que la période est de 30 jours (1 mois), numérotés $1, \dots, 30$, avec donc des jours :

31, 32, ..., 60, 61, 62, ..., 90, ... qui reproduisent les demandes L_i enregistrées aux jours $i = 1, \dots, 30$. La question posée est alors simple : Comment définir **une politique de production/stockage, c'est-à-dire une fonction d qui, à tout instant de base $i = 1, \dots, 30$, tout état $e = (e_S, e_M)$ associe une décision $d(i, e) = (d_P, d_M)$** appliquée de la même façon à tout instant réel $30 \times j + i$, $j = 0, \dots$, et qui permette :

- De satisfaire les demandes périodiques $L_i = L_{i+30 \times j}$;
- De minimiser le coût induit par période de 30 jours.

Ainsi formulé, notre énoncé laisse apparaître des zones de flou :

- Par exemple, la notion de périodicité s'applique-t-elle aux états ? En d'autres termes, l'état induit par une trajectoire périodique de notre système à un instant i doit-il être le même pour tous les instants de la forme $30 \times j + i$? Ou bien peut-on se contenter de ce que cette périodicité sur les états soit seulement un multiple de 30 ?
- Egalement, doit-on considérer que l'état courant rencontré quand $i = 1$ (ou 0) fait partie de la **trajectoire périodique** cherchée, ou bien doit-on considérer qu'il fait partie d'une **trajectoire transitoire** qui permettra d'atteindre une **trajectoire stationnaire périodique** ? Dans ce dernier cas, doit-on se préoccuper de la durée et du coût de cette trajectoire transitoire ?

On voit que le problème devient dès lors assez vite compliqué, avec la place pour de multiples variantes. Pour simplifier, on va d'abord supposer que :

- On exige une périodicité de 30 jours sur les états : pour tout i , l'état d'une trajectoire en $i + 30$ doit être le même que l'état en i ;
- On ne se préoccupe pas de l'état transitoire.

Dans ce cas, on voit que l'on cherche un circuit à 30 sommets et de longueur minimale dans un graphe orienté tel que représenté sur la Figure (3.6).

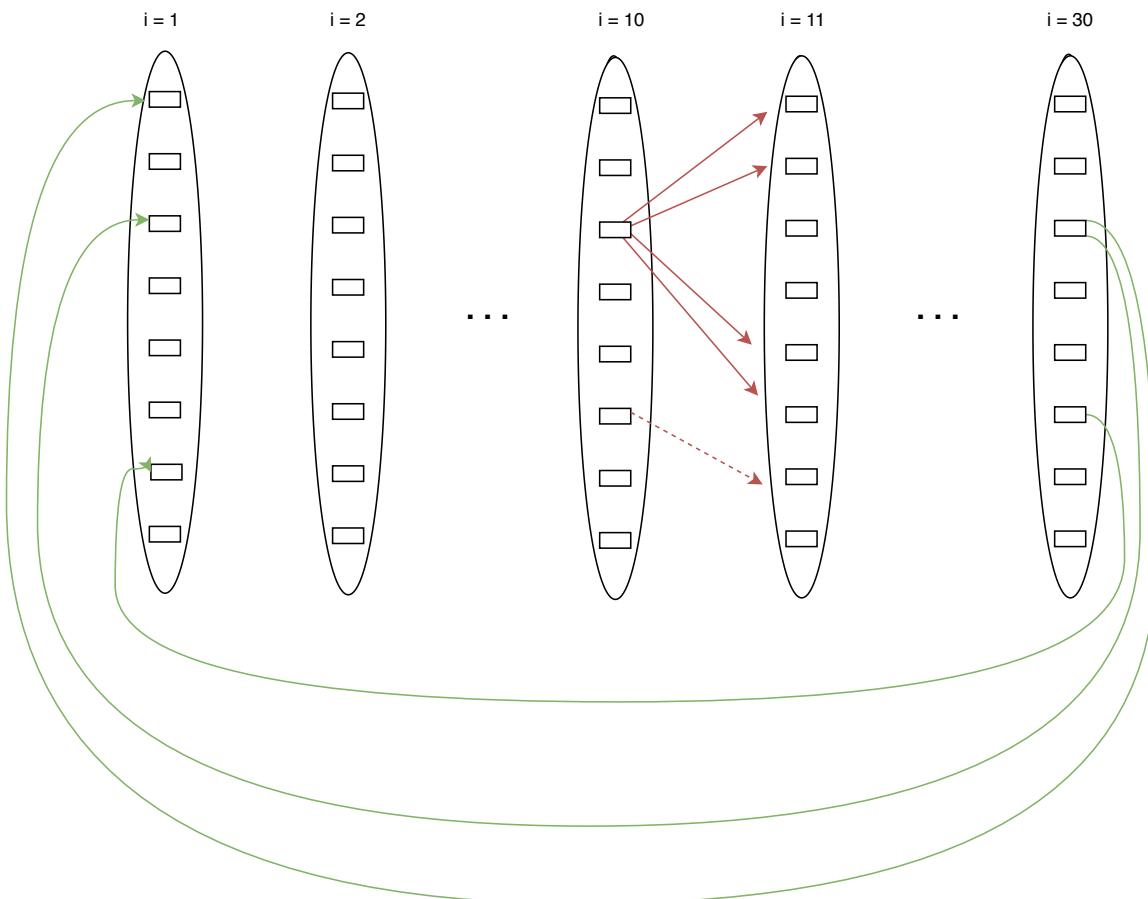


FIGURE 3.6 – Graphe des Etats Cycliques.

Une première approche vient comme suit :

1. On applique l'algorithme **PD** tel que vu en Section 3.3.1 à partir de tous les états initiaux $(0, e = (e_S, e_M))$ possibles, en considérant à chaque fois que l'état final est défini comme $(N + 1, e)$. On peut le faire en Backward afin de récupérer plus aisément les suites de décisions pour chaque état. On obtient alors pour chaque état f , et chaque i , une valeur $V_e(i, f)$

2. On choisit e de telle sorte que la valeur $V_e(N + 1, e)$ soit minimale. Les séquences de décisions issues de 1) nous fournissent alors notre stratégie.

Sans entrer dans le détail, on voit que cette façon de procéder présente deux inconvénients :

- Elle est coûteuse en temps, puisqu'il faut énumérer tous les états e possibles au jour $i = 1$, et leur appliquer le **PD** de la Section 3.3.1 ;
- Elle ne nous dit rien sur les états transitoires, c'est-à-dire ceux par lesquelles il faudra passer avant de stabiliser la trajectoire autour de sa version périodique optimale, alors que dans la pratique, les systèmes réels ne sont jamais vraiment stationnaires.

On peut dès lors appliquer une autre approche :

1. On choisit un (ou plusieurs états finaux « plausibles », correspondant au passage en $i = 30 \times j + 1$;
2. On remonte alors le processus **PD** vu à la section 3.3.1, en stratégie Backward, sur un temps suffisamment long (par exemple de $j = 10$ jusqu'à $j = 0$) ;
3. On considère alors les décisions correspondant à $j = 0$ comme fournissant les décisions en régime stationnaire ; On répertorie l'ensemble E_{Stat} des états correspondant à $E(0)$ selon le schéma ainsi appliqué ; Pour chaque état e on calcule son coût par période $CP(e)$;
4. Pour un état e_0 donné à un instant $i = 0$, on applique le processus **PD** en Forward à partir de $i = 0$ et e_0 , en considérant un état final fictif e_F associé à $i = 31$, et une transition fictive permettant de passer de tout couple $(e, 30)$, $e \in E_{Stat}(31, e_F)$, et dotée d'un coût $\beta \times CP(e)$, β étant un *scaling coefficient*, supposé pondérer les poids respectifs de la partie transitoire et de la partie stationnaire.

Cette approche suscite aussi des questions et des remarques :

- Comment choisir β , c'est-à-dire comment équilibrer transitoire et stationnaire ?
- La partie stationnaire de la trajectoire peut ne pas correspondre à 30, mais à un multiple de 30.

Dans cette section, nous avons fait un état de l'art de la programmation dynamique. Dans la section suivante, nous présenterons brièvement ce qu'est l'*apprentissage*.

3.4 Apprentissage automatique

L'intelligence artificielle abrégé IA est un concept général qui fait allusion à des systèmes ou machines qui imitent l'intelligence humaine. Le *Machine Learning* abrégé ML est un sous-domaine de l'intelligence artificielle qui apprend et améliore des performances en fonction des données traitées. Le *machine learning* apporte des solutions aux problèmes complexes en analysant de grande quantité de données. Les principales étapes d'un projet de *machine learning* sont les suivantes : la définition claire du problème à résoudre, l'acquisition des données, la préparation et le nettoyage des données, la séparation des données, le choix des procédés de *machine learning*, l'entraînement des procédés, l'évaluation et la validation des procédés, le déploiement et la mise en « production » des procédés pour résoudre le problème posé.

Dans cette section, nous commencerons par identifier les principales difficultés de l'apprentissage. Nous nous attarderons ensuite sur les réseaux de neurones car nous avons utilisé cette technique pour faire une approximation des coûts de notre problème. Nous finirons par présenter l'API Keras de Tensorflow Keras utilisé pour implémenter nos réseaux de neurones.

3.4.1 La problématique générale de l'apprentissage

La problématique de l'apprentissage est confrontée à plusieurs difficultés que nous présentons dans cette section. La problématique de l'apprentissage est dans la plupart des cas une problématique d'approximation. C'est une correspondance R qui à une entrée I associe une sortie complexe O , que l'on veut approcher via un procédé $I \rightarrow A^\lambda(I)$, où λ est le jeu de paramètres à apprendre, et où A^λ est un algorithme rapide. Le but est que l'erreur commise en remplaçant O par $A^\lambda(I)$ soit faible. Cette erreur pouvant être mesurée :

- Comme pourcentage de fois où O et $A^\lambda(I)$ sont différents, dans le cas où O désigne un identificateur de classe ou bien une propriété ;
- Comme une valeur moyenne de $(A^\lambda(I) - O)$ dans le cas où O désigne une quantité ou un vecteur de nombres.

Ce dernier point est notamment le cas pour ce qui est de notre étude, puisque l'on veut remplacer un algorithme complexe de programmation dynamique par un algorithme plus simple que l'on puisse utiliser comme estimateur de solution au sein d'un processus d'optimisation.

Les points durs de la mise en œuvre d'un projet d'apprentissage sont les suivants : le premier point est l'identification de la correspondance $I \rightarrow O$ que l'on veut soumettre à apprentissage, le deuxième point porte sur le choix du type de procédé A^λ dont on va se servir et le troisième point porte sur l'obtention de données.

Afin d'illustrer la difficulté d'identifier la correspondance $I \rightarrow O$ que l'on veut soumettre à apprentissage, on peut se référer au jeu des échecs : on veut utiliser l'apprentissage afin de construire un logiciel qui joue bien, et donc, qui, face à une position P donnée (un état de l'échiquier) choisisse le bon mouvement M à effectuer. Que faut-il alors chercher à apprendre : la correspondance $P \rightarrow M$, ou bien une correspondance auxiliaire $P \rightarrow Qualite(P)$ qui nous fournit une estimation de la qualité du jeu du point de vue du joueur machine ? L'expérience montre que seule la deuxième approche permettra d'obtenir de bons résultats.

Le procédé A^λ dont on va se servir peut être une formule portant sur des indicateurs caractéristiques de I ; il peut aussi être un système à base de règles (systèmes expert) ; il peut enfin être un automate relativement complexe tel qu'un réseau neuronal ou bien un automate SVM.

L'obtention de données, c'est-à-dire de couples (I, O) , où I est une instance et O un résultat théorique associé à I , est une tâche complexe. Il faut que l'on dispose de nettement plus de tels couples (I, O) que de paramètres λ à apprendre, et il faut veiller à ce que ces instances I soient suffisamment diversifiées. Ceci peut s'avérer difficile et pose la question de l'apprentissage supervisé : dans le cas supervisé, on dispose de suffisamment de couples (I, O) d'entraînement, et on ajuste le vecteur de paramètres λ grâce à ces instances ; dans le cas contraire, il faut faire en sorte que le procédé A^λ apprenne lui-même le vecteur λ , en corrigeant de lui-même ses erreurs. On parle alors d'apprentissage non supervisé : autoapprentissage, apprentissage par renforcement, etc. Mettre en œuvre un procédé d'apprentissage non supervisé est dans tous les cas quelque chose de difficile.

3.4.2 Les réseaux de neurones artificiels

Les réseaux de neurones artificiels cherchent à simuler le comportement des neurones du cerveau humain [71]. Ce sont des intelligences artificielles qui donnent aux ordinateurs la capacité d'« apprendre à apprendre » et par la suite, à agir et réagir comme les humains, en améliorant leur mode d'apprentissage et leurs connaissances de façon autonome sur la durée. Les réseaux de neurones ont

une pléthore d'applications parmi lesquelles on peut citer : le traitement d'images (la reconnaissance faciale, la détection d'objets), le traitement de la langue et le traitement du signal.

Description générale et représentation graphique d'un réseau de neurones

Un réseau de neurones est généralement représenté comme une succession de « couches ». Chaque couche contient 1 ou plusieurs neurones. Les neurones d'une couche sont reliés aux neurones de la couche suivante de manière complète, voir figure (3.7), ou partielle, voir figure (3.8).

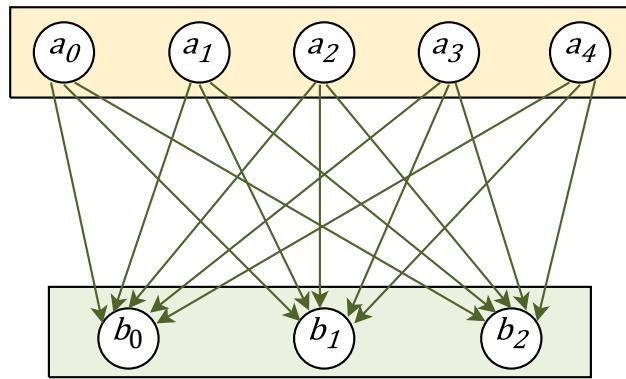


FIGURE 3.7 – Représentation d'une couche de 5 neurones **complètement** reliée à une couche de 3 neurones.

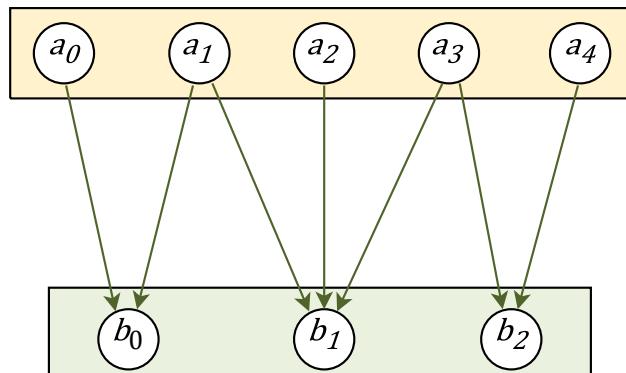


FIGURE 3.8 – Représentation d'une couche de 5 neurones **partiellement** reliée à une couche de 3 neurones.

Chaque neurone j a une valeur d'entrée $in(j)$ (aussi appelée valeur de préactivation) et une valeur de sortie $out(j)$ (aussi appelée valeur d'activation). Dans le cas où la fonction d'agrégation est la somme pondérée, la valeur d'entrée d'un neurone est égale à la somme des valeurs de sortie des neurones le précédent, pondérée par des poids, à laquelle on ajoute un biais. Les poids sont souvent représentés sur les arcs et le biais est souvent représenté comme un neurone supplémentaire, de valeur de sortie 1 (fixe) avec un poids B , voir figure (3.9). La valeur de sortie d'un neurone est égale à sa valeur d'entrée à laquelle on applique une fonction dite d'activation. La figure (3.10) donne les valeurs d'entrée et de sortie du neurone b de la figure (3.9) avec une fonction d'activation ϕ .

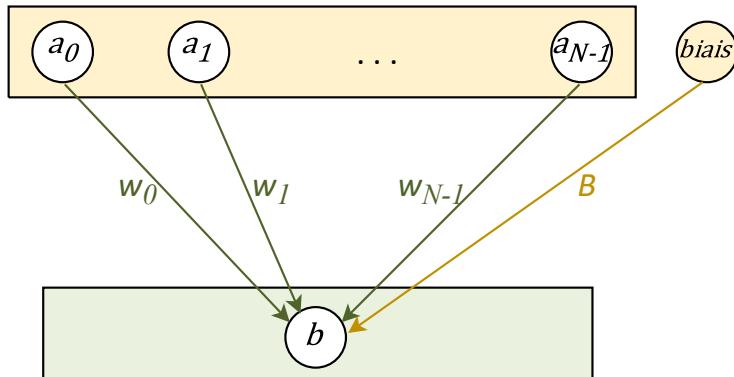


FIGURE 3.9 – Représentation des poids et du biais.

$$\begin{aligned}
 in(b) &= B + \sum_{i=0}^{N-1} out(a_i).w_i \\
 &\downarrow \\
 &b \\
 &\downarrow \\
 out(b) &= \phi(in(b)) = \phi\left(B + \sum_{i=0}^{N-1} out(a_i).w_i\right)
 \end{aligned}$$

FIGURE 3.10 – Calcul des valeurs d'entrée et de sortie du neurone b du réseau de la Figure (3.9).

Il existe plusieurs types de fonction d'activation parmi lesquelles on peut citer :

- La fonction linéaire : $f(x) = x$
- La fonction sigmoïde : $f(x) = \frac{1}{1 + e^{-x}}$
- La fonction RELU :

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0. \end{cases}$$

- La fonction softmax : $f(x)_j = \frac{e^{x_j}}{\sum_{k=1, \dots, K} e^{x_k}}$ pour tout $j \in \{1, \dots, K\}$, dans ce cas $\sum_j f(x)_j = 1$.

Dans le cas général, il y a plusieurs neurones par couche et les poids utilisés pour calculer la valeur d'entrée d'un neurone sont différents d'un neurone à l'autre, ainsi que les biais, voir Figure (3.11). On a pour habitude de regrouper ces poids dans une matrice, voir Figure (3.12).

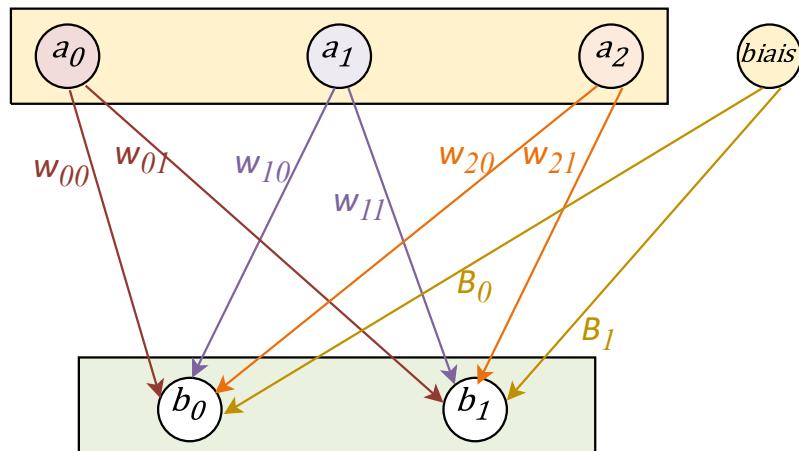


FIGURE 3.11 – Poids et biais entre deux couches de neurones non unitaires.

$$\begin{array}{ccc}
 & b_0 & b_1 \\
 & \downarrow & \downarrow \\
 a_0 & \xrightarrow{\quad} & \begin{matrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{matrix} \\
 a_1 & \xrightarrow{W = } & \\
 a_2 & \xrightarrow{\quad} &
 \end{array}$$

$$B = (B_0, B_1)$$

FIGURE 3.12 – Matrice des poids W et vecteur des biais B associés à la Figure (3.11).

Description du fonctionnement des réseaux de neurones

Le fonctionnement général du processus d'apprentissage est :

- Pour une entrée I , on calcule l'objet de sortie $A^\lambda(I) = O^\lambda$, calculée par le réseau A^λ (λ désigne le vecteur de coefficients synaptique et de biais à apprendre) ; Puis on évalue l'erreur $Err(O^\lambda)$ induite par rapport à un résultat souhaité ; Puis on ajuste λ afin de diminuer cette erreur en appliquant une formule $\lambda \leftarrow \lambda - \delta.Grad(Err(O^\lambda))$, où δ est une petite valeur positive, qu'on nomme le pas.
- On applique ce processus pour un ensemble d'instances I (ou de paquets d'instances) dites d'entraînement, générées aléatoirement. Le processus global ainsi induit est dit de **Gradient Stochastique**. C'est parce que l'on a besoin de pouvoir calculer cette quantité gradient $Grad(Err(O^\lambda))$ que l'on utilise des coefficients synaptiques et des impulsions continues, ainsi que des fonctions d'activation différentiables.

Difficultés liées à la mise en œuvre de réseau de neurones

Les difficultés liées à la mise en œuvre des réseaux de neurones sont :

- Une première difficulté tient à la conception du réseau : il n'existe à ce propos pas de méthode clairement reconnue. L'intuition est toutefois d'essayer d'épouser la logique du fonctionnement de la correspondance $I \rightarrow O$ que l'on cherche à apprendre.
- Une deuxième difficulté majeure tient au fait qu'un réseau neuronal tend à mettre en jeu un nombre important de coefficients à apprendre, ce qui signifie qu'il faut aussi un nombre important d'instances I pour les phases d'entraînement et de validation (à priori sensiblement plus que de coefficients synaptiques). Obtenir ces instances en même temps que leur résultat souhaité O peut constituer une vraie difficulté. Si le nombre d'instances est insuffisant, alors on risque de bien réussir la phase entraînement, mais d'avoir un échec sur la phase validation. Une difficulté en apprentissage est le sur-apprentissage qui est cette tendance qu'à un procédé à faire de bonnes prédictions uniquement sur les données d'entraînement et à faire de mauvaises prédictions sur de nouvelles données.

3.4.3 L'API Keras de Tensorflow Keras

Dans cette section, on note Tensorflow Keras de manière abrégée : tf.keras. De plus, nous utiliserons le vocabulaire suivant :

- le terme **couche** sera utilisé pour la description générale d'un réseau alors que le terme **layer** sera utilisé pour faire référence à une couche particulière prédéfinie de tf.keras ;
- les termes **modèle** et **réseau** seront utilisés indistinctement (modèle étant le vocabulaire utilisé dans tf.keras pour faire référence à un réseau de neurones).

Les couches et les poids dans tf.keras

Dans tf.keras il existe différentes couches, appelées layers, que l'on peut utiliser facilement pour créer un réseau de neurones (également appelé modèle). Ces layers définissent la manière dont les neurones reçoivent l'information des layers précédents.

- **Le layer Dense**

Le layer Dense est le layer de référence pour créer des couches dont les neurones reçoivent en entrée une valeur égale à la somme pondérée de l'ensemble des neurones du layer précédent (comme dans la Figure (3.11)). Le layer Dense permet donc de créer une couche qui est reliée de manière complète à la couche précédente. Le code suivant montre comment créer le réseau de la Figure (3.11).

```

1 #création de la couche des inputs (avec 3 neurones , car les entrées sont de taille
  3)
2 my_input = keras.Input(shape=(3,) , name="input")
3
4 # création de la couche de sortie avec 2 neurones
5 sortie = keras.layers.Dense(2, activation='linear' , name="sortie")(my_input)
6
7 #construction du modèle
8 model = keras.Model(inputs=my_input, outputs=sortie)
```

Dans tf.keras, les couches sont représentées par une matrice des poids et un vecteur des biais. La fonction suivante permet d'afficher les poids d'une couche. Elle prend en paramètre le modèle et le nom de la couche dont on souhaite afficher les poids.

```

1 def affichePoids(model, name):
2
3     print ("affichage des poids de la couche", name)
4
5     #afficher les poids qui arrivent sur une couche
6     WS = model.get_layer(name).get_weights()
7
8     print("weights = ", WS[0] )
9     print("biais = ", WS[1] )

```

Par exemple, si on souhaite afficher les poids de la couche de sortie du modèle Dense précédent il faut écrire `affichePoids(model, "sortie")`. Les poids sont alors affichés comme à la figure (3.13). La représentation graphique de ces poids est à la figure (3.14).

```

affichage des poids de la couche sortie
weights = [[1.          4.]
            [2.          5.]
            [3.          6.]]
biais = [10.        20.]

```

FIGURE 3.13 – Affichage des poids.

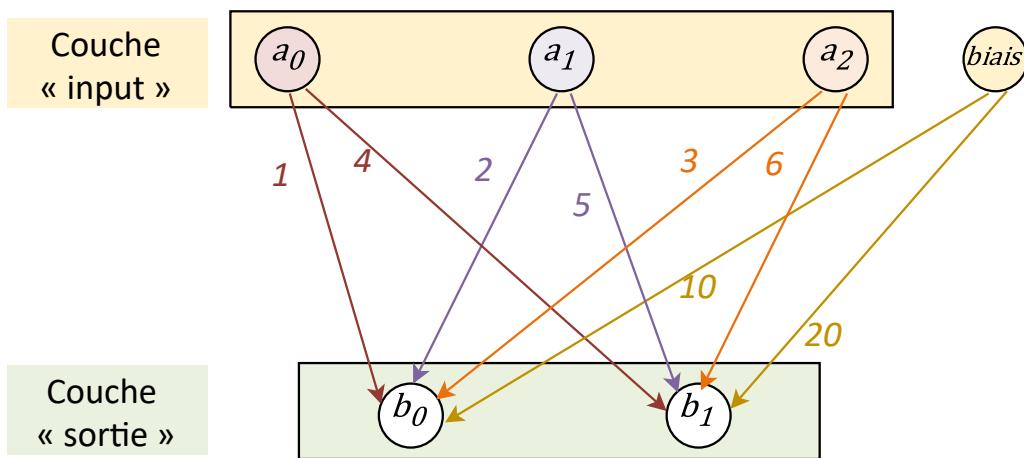


FIGURE 3.14 – Représentation graphique des poids associés à la Figure (3.13).

Ils signifient que les valeurs d'entrée (ou de préactivation) pour les 2 neurones (b_0, b_1) de la couche de sortie sont calculées de la manière suivante (pour simplifier les notations $out(a)$ est remplacé par a) : $in(b_0) = a_0 + 2a_1 + 3a_2 + 10$, $in(b_1) = 4a_0 + 5a_1 + 6a_2 + 20$.

Initialisation des poids du layer Dense

L'initialisation des poids est un élément important qui joue un rôle clé dans la descente du gradient. Par défaut cette initialisation est réalisée par tf.keras de manière aléatoire. Cependant, si on a une idée des poids finaux, il peut être intéressant de réaliser cette initialisation nous-même. Il suffit pour cela de passer en paramètre la matrice des poids et le vecteur des biais lors de la création du layer Dense. Le code suivant montre comment construire le réseau de la Figure (3.14) en initialisant les poids (y compris les biais) à la main.

```

1 # création de la couche des inputs (avec 3 neurones)
2 my_input = keras.Input(shape=(3,), name="input")
3
4 # création de la couche de sortie avec un 2 neurones et initialisation des poids
5 my_biais = np.array([10, 20])
6 my_poids = np.array([[1, 4], [2, 5], [3, 6]])
7 sortie = keras.layers.Dense(2, activation='linear', weights=[my_poids, my_biais],
     name="sortie")(my_input)
8
9 # construction du modèle
10 model = keras.Model(inputs=my_input, outputs=sortie)
11
12 # dessin du modèle
13 keras.utils.plot_model(model, "Func_tutoPoids2.png", show_shapes=True, dpi=192)
```

□ Différentes manières de fusionner les couches : *Merging layers*

tf.keras offre la possibilité de fusionner, de différentes manières, une ou plusieurs couches d'un réseau. Pour cela il faut utiliser des layers particuliers appelés merging layers.

— Concaténation de plusieurs couches

La manière la plus simple de fusionner des couches est de les concaténer : dans ce cas les neurones sont simplement recopier de leur couche initiale vers la couche de concaténation sans modification de leur valeur de sortie. Le code suivant montre comment concaténer 3 couches à l'aide du layer *Concatenate*.

```

1 # création de 3 couches des entrées de taille respective 3, 2 et 4
2 my_input1 = keras.Input(shape=(3,), name="input1")
3 my_input2 = keras.Input(shape=(2,), name="input2")
4 my_input3 = keras.Input(shape=(4,), name="input3")
5
6 # concaténation des 3 couches des entrées
7 concat = keras.layers.concatenate([my_input1, my_input2, my_input3])
8
9 # création de la couche de sortie avec un 2 neurones
10 sortie = keras.layers.Dense(2, activation='linear', name="sortie")(concat)
11
12 # construction du modèle
13 model = keras.Model(inputs=[my_input1, my_input2, my_input3], outputs=sortie)
```

— Multiplication terme à terme de deux couches

Un autre layer très intéressant est le layer *Multiply*. Il permet de multiplier la valeur de sortie de 2 couches de neurones terme à terme. L'extrait de code suivant illustre son utilisation.

```

1 # création de 2 couches d'entrée de taille 3
2 my_input1 = keras.Input(shape=(3,), name="input1")
3 my_input2 = keras.Input(shape=(3,), name="input2")
4
5 # une couche cachée par couche d'entrée
6 couche1 = keras.layers.Dense(3, activation='relu', name="C1")(my_input1)
7 couche2 = keras.layers.Dense(3, activation='relu', name="C2")(my_input2)
8
9 # multiplication des 2 couches précédentes
10 sortie = keras.layers.Multiply()([couche1, couche2])

```

— Afficher la sortie des couches internes d'un modèle

Afin de s'assurer que les couches ont été fusionnées comme on l'entend, nous avons besoin d'accéder à la sortie des couches internes d'un modèle. Pour cela il faut :

1. Récupérer la couche du modèle concernée à l'aide de son nom (il faut donc lui avoir donné un nom lors de sa création),
2. Appeler la couche avec les inputs du modèle pour lesquels on souhaite connaître la sortie.

Le code suivant illustre ceci sur le modèle précédemment créé.

```

1 # exemple de données pour les 2 couches input
2 in1_test = np.asarray([[1,2,3]])
3 in2_test = np.asarray([[4,5,6]])
4
5 # on récupère les couches internes à partir de leur nom
6 c1_layer = keras.Model(inputs=model.input, outputs=model.get_layer('C1').output)
7 c2_layer = keras.Model(inputs=model.input, outputs=model.get_layer('C2').output)
8
9 # on donne aux couches internes les inputs souhaités
10 couche1_out = c1_layer({'input1': in1_test, 'input2': in2_test})
11 couche2_out = c2_layer({'input1': in1_test, 'input2': in2_test})
12
13 # on récupère la sortie du modèle avec les inputs souhaités
14 sortie_out = model.predict({'input1': in1_test, 'input2': in2_test})
15
16 # on affiche la sortie des différentes couches
17 print('couche1 : ', couche1_out)
18 print('couche2 : ', couche2_out)
19 print('sortie : ', sortie_out)

```

On obtient, par exemple, l'affichage suivant :

```

couche1 : tf.Tensor([-1.399934 -2.0355947 3.6685464], shape=(1, 3), dtype=float32)
couche2 : tf.Tensor([0.44896154 0.15367174], shape=(1, 3), dtype=float32)
sortie : [[-0.44896154 0.15367174]]

```

On constate que la couche de sortie est bien la multiplication terme à terme des couches internes 1 et 2.

NB. Si vous testez ce code vous n'obtiendrez probablement pas la même chose, les poids étant initialisés de manière aléatoire.

Relier deux couches de neurones de manière partielle

Problématique générale

Il n'existe pas, à notre connaissance, de layer prédefini qui permette de relier 2 couches de manière partielle (c'est-à-dire telle que présentée dans la Figure (3.8)).

Une solution possible serait de créer son propre layer (ce qui est réalisable dans tf.keras en créant une classe dérivée de la classe Layer). Cependant cette technique demande une très bonne compréhension des différents éléments qui forment un layer au sens de tf.keras ainsi que de la manière dont ils s'articulent. Elle semble, par conséquent, assez fastidieuse à mettre en place.

Une autre solution, qui est celle que nous avons choisis de présenter ici, consiste à utiliser les layers de tf.keras prédefinis et à les combiner de manière à obtenir exactement la topologie de réseau souhaitée. C'est l'objet de la prochaine sous-section.

Combiner des layers prédefinis de tf.keras pour relier 2 couches de manière partielle

Remarque 1 Dans cette section, on ne s'intéresse qu'aux arcs (et donc qu'aux poids) entre 2 couches de neurones. Le biais n'est pas pris en compte, et, même s'il existe, on ne le représente pas sur les schémas afin de ne pas surcharger les illustrations.

— Principe général

L'idée générale est la suivante : supposons que l'on souhaite créer une couche A de N neurones (a_0, \dots, a_{N-1}) qui envoie de l'information à une couche B de M neurones (b_0, \dots, b_{M-1}).

De plus nous souhaitons choisir exactement quel neurone de la couche A envoie de l'information à quel neurone de la couche B (Figure (3.15)). Nous avons donc besoin de pouvoir manipuler chaque neurone de manière indépendante. Nous réalisons cela en 4 étapes (voir (Figure (3.16)) :

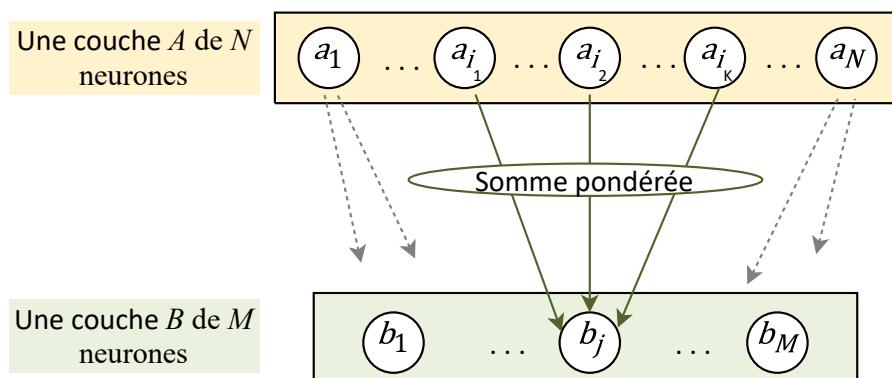


FIGURE 3.15 – Représentation d'un réseau avec 2 couches partiellement reliées.

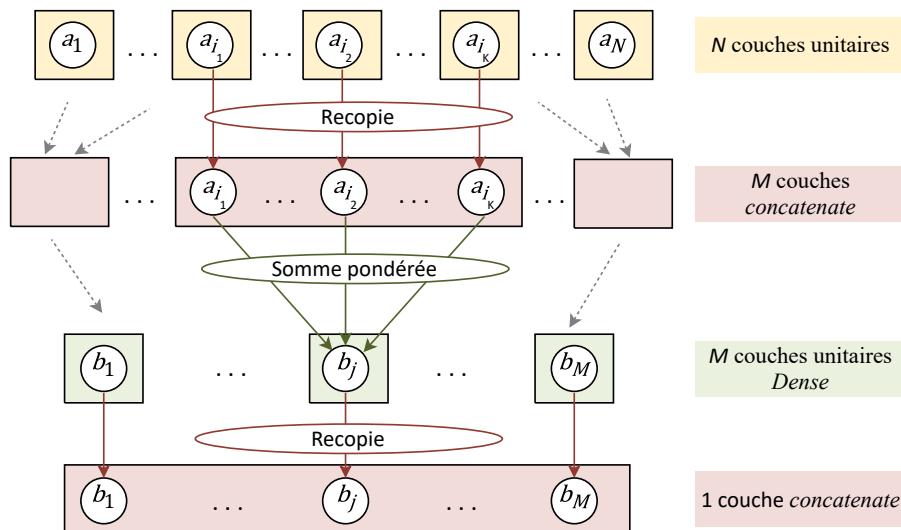


FIGURE 3.16 – Représentation du réseau de la figure (3.15) en utilisant exclusivement des layers tf.keras Dense et Concatenate.

1. Créer N « sous-couches » unitaires A_0, \dots, A_{N-1} au lieu d'une couche de N neurones : la sous-couche $A_{i,(i=0\dots N-1)}$ contient uniquement le neurone a_i .
2. Pour chaque neurone $b_{j,(j=0\dots M-1)}$ de la couche B , identifier les neurones a_{i_1}, \dots, a_{i_K} ($0 \leq i_1 \leq i_K \leq N - 1$) qui envoient de l'information à b_j . Concaténer les sous-couches A_{i_1}, \dots, A_{i_K} correspondantes. On obtient ainsi M nouvelles sous-couches C_0, \dots, C_{M-1} de type Concatenate. Notons qu'un même neurone $a_{i,(i=0\dots N-1)}$ peut se retrouver dans plusieurs sous-couches Concatenate (sa valeur sera la même dans toutes les sous-couches).
3. Créer M sous-couches unitaires (B_0, \dots, B_{M-1}) telles qu'une sous-couche $B_{j,(j=0\dots M-1)}$ contient uniquement le neurone b_j et reçoit en entrée la sortie de la sous-couche C_j .
4. Pour finir, concaténer les sous-couches unitaires (B_0, \dots, B_{M-1}) de sorte de retrouver une couche de M neurones (b_0, \dots, b_{M-1}).

Remarque 2 Les entiers N et M peuvent être potentiellement grands et/ou inconnus au moment où on écrit le code (par exemple si on les lit dans un fichier). C'est pourquoi il semble judicieux d'utiliser des tableaux pour stocker les différentes sous-couches.

La sous-section suivante illustre ceci sur un exemple.

— Illustration sur un exemple

On souhaite représenter le réseau de la Figure (3.8) en utilisant uniquement des layers Dense et Concatenate et en suivant la méthodologie donnée en section 3.2.1.

Le résultat obtenu est représenté Figure (3.17).

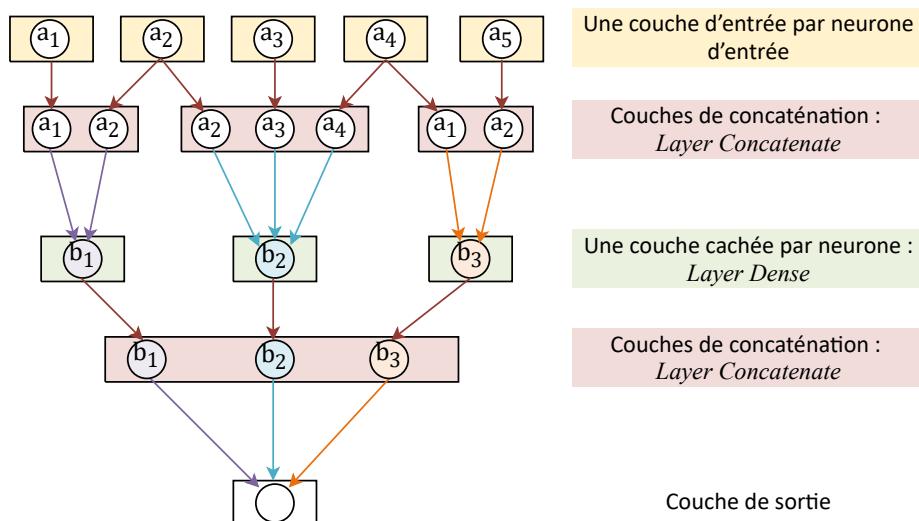


FIGURE 3.17 – Le réseau de la figure (3.8) représenté avec des layers Dense et Concatenate.

Le code qui permet de programmer ce réseau est donné ci-dessous.

```

1 # ETAPE 2.1 : création des sous-couches A_0 ... A_{N-1} (ici N = 5)
2
3 N = 5
4 ssCoucheA = []
5
6 for i in range(N):
7     tmp = keras.Input(shape=(1,), name="input"+str(i))
8     ssCoucheA.append(tmp)
9
10 # ETAPE 2.2 : on crée les M = 3 couches de concaténation
11
12 ssCoucheC = []
13
14 ssCoucheC.append(keras.layers.concatenate([ssCoucheA[0], ssCoucheA[1]]))
15 ssCoucheC.append(keras.layers.concatenate([ssCoucheA[1], ssCoucheA[2], ssCoucheA[3]]))
16 ssCoucheC.append(keras.layers.concatenate([ssCoucheA[3], ssCoucheA[4]]))
17
18 # ETAPE 2.3 : création des sous-couches B_0 ... B_{M-1} (ici M = 3)
19 ssCoucheB = []
20 M = len(ssCoucheC)
21 for i in range(M):
22     ssCoucheB.append(keras.layers.Dense(1, activation='linear')(ssCoucheC[i]))
23
24 # ETAPE 2.4 : on concatene les sous-couches B
25 sortie = keras.layers.concatenate(ssCoucheB)
26
27 # construction du modèle : définition des couches d'entrée et de sortie
28 model = keras.Model(inputs=ssCoucheA, outputs=sortie)
29
30 # dessin du modèle
31 keras.utils.plot_model(model, "reseauFunctionalSparse.png", show_shapes=True,
32                         dpi=192)
```

Fixer des poids dépendant des instances

Le but de cette section est de montrer comment on peut prendre en compte, à l'intérieur d'un réseau, et de manière explicite, des valeurs propres à chaque instance. Pour fixer les idées, nous allons illustrer notre propos sur le réseau de la Figure (3.18). Dans cet exemple nous supposons qu'une entrée est formée de deux types de valeurs : N valeurs a_0, a_1, \dots, a_{N-1} M coefficients $\lambda_0, \lambda_1, \dots, \lambda_{M-1}$

Les valeurs a_0, a_1, \dots, a_{N-1} sont les entrées du réseau alors que les coefficients $\lambda_0, \lambda_1, \dots, \lambda_{M-1}$ sont utilisés uniquement pour calculer la valeur de sortie du réseau s qui est telle que :

$$s = \sum_{j=0}^{M-1} \lambda_j \times \text{out}(b_j)$$

où $b_{j,(j=0,\dots,M-1)}$ est le j^{eme} neurone de la couche précédant la sortie du réseau (voir Figure (3.18))

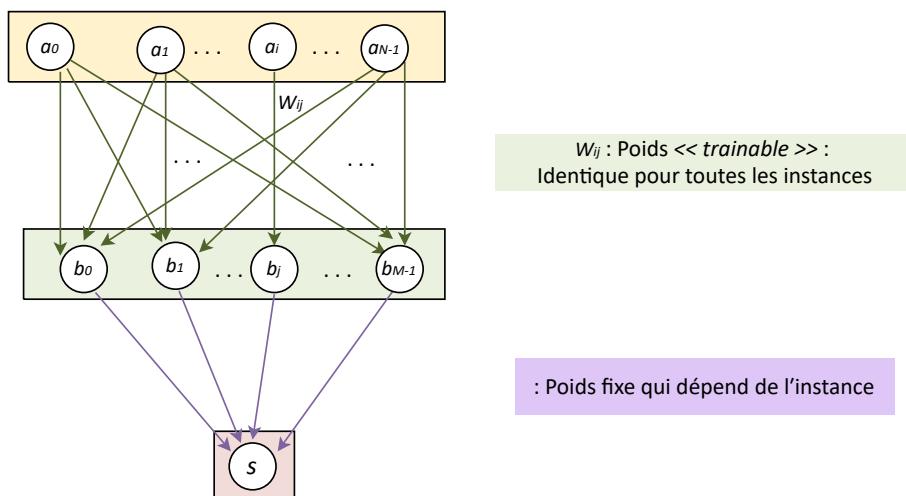


FIGURE 3.18 – Exemple de réseau dans lequel des poids fixes et dépendant de l'instance interviennent.

Nous avons vu dans comment fixer les poids d'un layer Dense. Malheureusement, cette technique ne fonctionne que pour les poids « classiques » : c'est-à-dire qui sont les mêmes pour toutes les instances. Pour personnaliser les poids en fonction de chaque instance il faut les donner au réseau comme une entrée (c'est-à-dire dans un layer Input) puis utiliser des layers spéciaux (par exemple le layer Multiply) qui permettent de combiner différentes couches entre-elles. Pour programmer le réseau de la figure (3.18), il faut donc le transformer en utilisant par exemple des layers Dense et Multiply, comme illustré dans la figure (3.19).

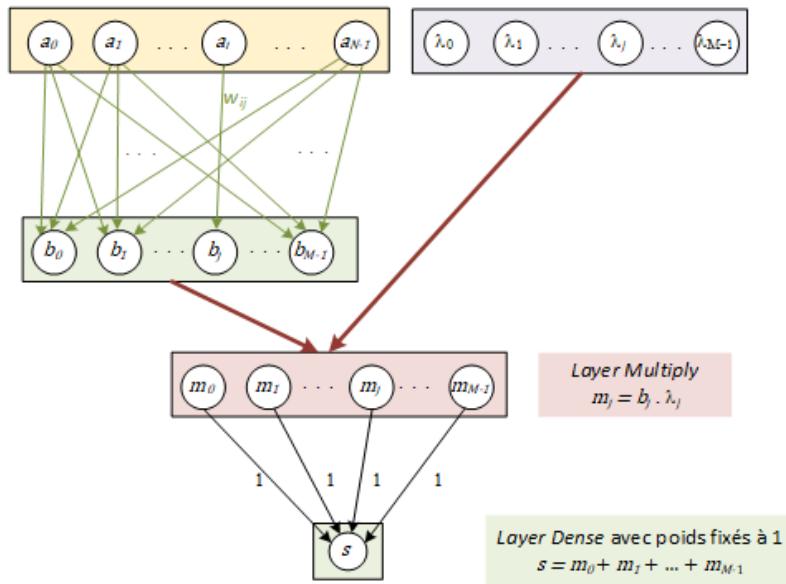


FIGURE 3.19 – Figure (3.18) reformulé avec des layers Dense et Multiply.

L'extrait de code suivant montre comment programmer le réseau de la Figure (3.19).

```

1 N = 8
2 M = 5
3
4 # création de 2 couches d'entrée de taille respective N et M
5 my_input1 = keras.Input(shape=(N,), name='input1')
6 my_input2 = keras.Input(shape=(M,), name='input2')
7
8 # une couche cachée
9 couche1 = keras.layers.Dense(M, activation='linear', name='C1')(my_input1)
10
11 # multiplication de la couche d'entrée 'input2' et de la couche cachée
12 mult = keras.layers.Multiply()([my_input2, couche1])
13
14 # 1 neurone de sortie = somme des sorties de la couche précédente
15 # on a donc des fixes poids = 1 et un biais = 0
16 my_biais = np.asarray([0])
17 my_poids = np.ones((M,1))
18 sortie = keras.layers.Dense(1, activation='linear', name='out', weights=[my_poids,
    my_biais], trainable=False)(mult)
19
20 # création du modèle
21 model = keras.Model(inputs=[my_input1, my_input2], outputs=sortie)

```

3.5 Conclusion

Ce chapitre était consacré à l'établissement de l'état de l'art des méthodes qu'on a exploré pour tenter de résoudre le problème traité dans le cadre de cette thèse. On a d'une part décrit ce qu'est la programmation dynamique et d'autre part décrit ce qu'est l'apprentissage. Le chapitre suivant sera consacré à la présentation de nos modèles.

CHAPITRE 4

PROGRAMMATION LINÉAIRE EN NOMBRES ENTIERS MIXTES DE SMEPC

Sommaire

4.1	Introduction	79
4.2	Le problème SMEPC : <i>Synchronous Management of Energy Production and Consumption</i>	79
4.3	Formulation mathématique	85
4.3.1	Variables	85
4.3.2	Fonction objectif	86
4.3.3	Contraintes du problème PM	86
4.3.4	Contraintes du problème VD	87
4.3.5	Contraintes de synchronisation	88
4.4	Formulation par programmation linéaire en variables mixtes : $MILP_{SMEPC}$	88
4.5	Relaxation linéaire de la formulation $MILP_{SMEPC}$ et contraintes additionnelles STC et EC : $RMILP_{SMEPC}$	89
4.5.1	Contraintes additionnelles EC et STC	90
	Contraintes STC : <i>Simple Time Constraints</i>	91
	Contraintes EC : <i>Energy Constraints</i>	91
4.6	Complexité du modèle SMEPC	93
4.7	Expérimentations numériques	93
4.7.1	Objectifs et contexte technique	93
4.7.2	Procédés de génération d'instances du SMEPC	93
	Génération du paquet d'instances INST_VAR	95
	Génération du paquet d'instances INST_CTE	95
4.7.3	Caractéristiques des instances	96
	Caractéristiques des instances du paquet INST_VAR	97
	Caractéristiques des instances du paquet INST_CTE	97
4.7.4	Caractéristiques des solutions	98
	Caractéristiques des solutions des instances du paquet INST_VAR	99
	Caractéristiques des solutions des instances du paquet INST_CTE	99
4.7.5	Résultats du modèle $MILP_{SMEPC}$ avec temps limite de 3 heures et 8 threads	100
	Résultats du modèle $MILP_{SMEPC}$ pour les instances du paquet INST_VAR	100
	Résultats du modèle $MILP_{SMEPC}$ pour les instances du paquet INST_CTE	101

4.7.6 Résultats des modèles $RMILP_{SMEPC}$ et $MILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread	102
Résultats du $RMILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread	102
Résultats du $MILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread	107
Observations et discussions	111
4.8 Conclusion	112
4.9 Annexes	112
4.9.1 Démonstration du théorème 2	112
4.9.2 Démonstration du lemme 2	113
4.9.3 Démonstration du théorème 3	113

4.1 Introduction

Ce chapitre présente la modélisation linéaire en nombres entiers qu'on a réalisé du problème SMEPC. Dans la **section 4.2**, on explique la problématique des travaux de recherche réalisés durant ce projet de thèse, à savoir la conception d'algorithmes et de modèles pour résoudre le problème de synchronisation de la production et des activités d'un véhicule autonome. On définit les données d'entrées et une solution d'une instance du problème **SMEPC**. La **section 8.2** liste les différentes extensions du problème **SMEPC**, qui seront considérées comme des perspectives éventuelles pour une extension de ce travail. La **section 4.3** présente le modèle mathématique du problème **SMEPC** en présentant les variables, la fonction objectif et les contraintes du problème. La **section 4.4** présente le programme linéaire à variable mixte de SMEPC. Dans la **section 4.5**, on présente la relaxation linéaire du programme précédent. La **section 4.6** présente les résultats de complexité qu'on a obtenu, puis, on démontre que le problème **SMEPC** est NP-difficile. On finit ce chapitre en présentant à la section 4.7 les résultats expérimentaux.

4.2 Le problème SMEPC : Synchronous Management of Energy Production and Consumption

Dans cette partie, on explique le problème **SMEPC** : Synchronous Management of Energy Production and Consumption. Pour cela, on précise les données d'entrées de notre problème. Ensuite, on présente ce qu'est une solution d'une instance du problème **SMEPC**. Enfin, on explique les sous-problèmes qui émanent du problème **SMEPC**.

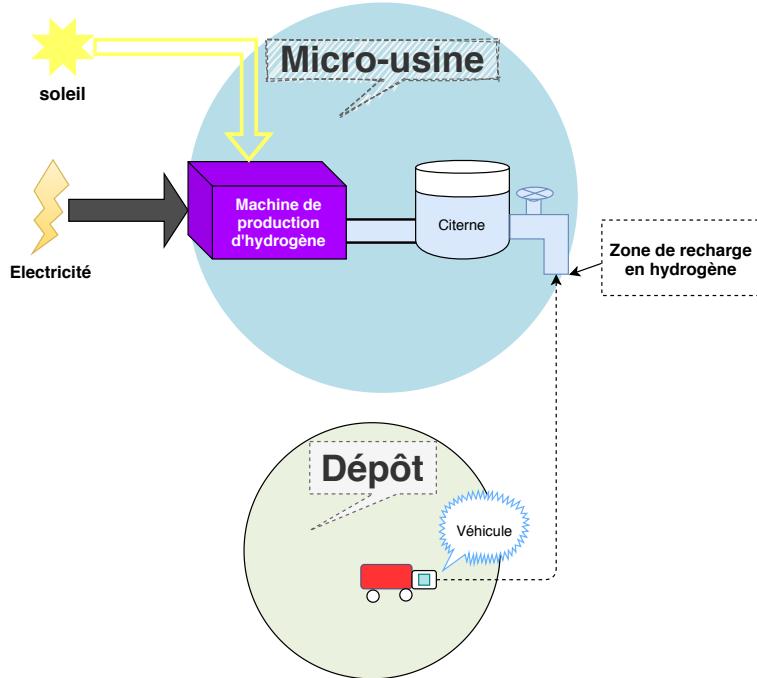


FIGURE 4.1 – Le dépôt et la micro-usine.

Nous considérons ici un véhicule qui doit effectuer des tâches de logistique interne, tout en suivant un itinéraire Γ qui commence et se termine sur une station particulière appelée *Dépôt* illustré à la figure (4.1). Le véhicule parcourt les stations $j = 1, \dots, M$, selon cet ordre. Cet itinéraire est appelé

une tournée. La station de départ *Dépôt* a l'étiquette $Depot = 0$ et est appelé dépôt initial. La station de fin *Dépôt* a l'étiquette $Depot = M + 1$ et est appelé dépôt final. Le temps nécessaire au véhicule pour se déplacer de la station j à la station $j + 1$ est égal à t_j , en tenant compte du temps passé par le véhicule pour effectuer des tâches locales aux stations. Le véhicule quitte le dépôt initial à la date 0 et doit terminer son parcours au plus tard à une date limite $TMax$.

Notre véhicule est alimenté à l'hydrogène (H^2). La capacité du réservoir du véhicule est appelé C^{Veh} et on connaît, pour tout $j = 0, \dots, M$, la quantité d'hydrogène e_j dont le véhicule a besoin pour aller de la station j à la station $j + 1$. La quantité d'hydrogène initiale dans le réservoir du véhicule est noté E_0 et le véhicule doit finir sa tournée avec au moins la quantité d'hydrogène E_0 , pour éviter les tournées triviales, c'est-à-dire les tournées durant lesquelles il n'y a ni recharge, ni production. Le véhicule doit se recharger périodiquement en hydrogène. Les opérations de recharges en hydrogène ont lieu dans une micro-usine (voir figure (4.1)), **proche du dépôt** : le temps nécessaire au véhicule pour se déplacer de la station j à la micro-usine est désigné par d_j , et le temps nécessaire au véhicule pour se déplacer de la micro-usine à la station j est noté d_{j+1}^* . De la même manière, l'énergie nécessaire au véhicule pour se déplacer de la station j à la micro-usine est désigné par ε_j , et l'énergie nécessaire au véhicule pour se déplacer de la micro-usine à la station j est noté ε_{j+1}^* . Les quantités t_j , d_j et d_{j+1}^* , ainsi que les quantités e_j , ε_j et ε_{j+1}^* sont non nulles et satisfont l'inégalité triangulaire (4.2). La quantité d'énergie initiale du véhicule doit pouvoir assurer son déplacement à la micro-usine $E_0 \geq \varepsilon_0$. La micro-usine et le dépôt ne se trouvent pas au même endroit car dans la réalité, il faudra toujours réaliser un trajet pour se déplacer entre ces deux lieux.

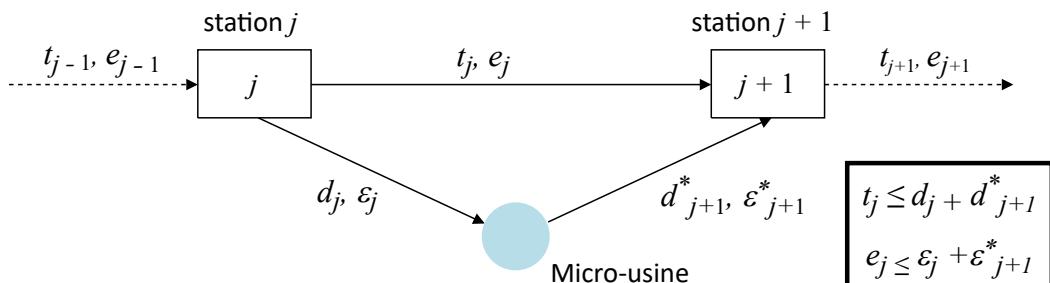


FIGURE 4.2 – Notations concernant les valeurs de temps et d'énergie pour deux stations consécutives j et $j + 1$ ($j = 0, \dots, M$).

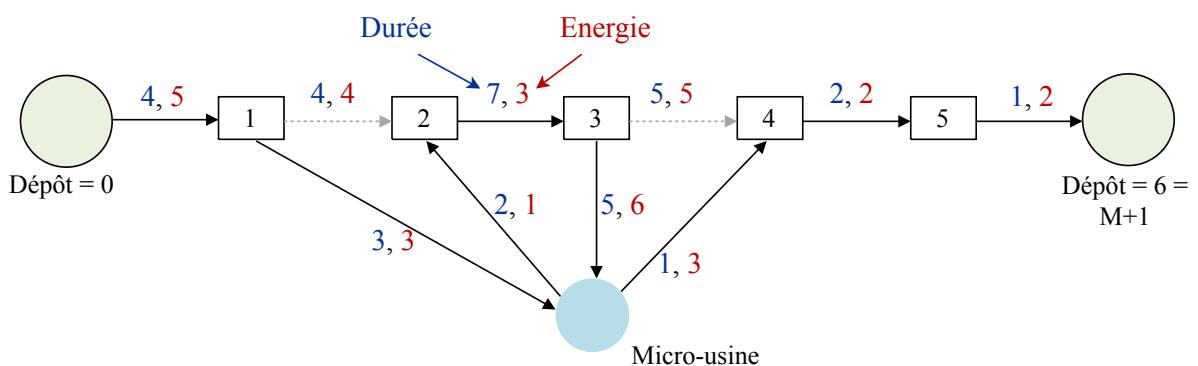


FIGURE 4.3 – Une tournée de véhicule avec ses opérations de recharges en hydrogène avec $M = 5$.

La figure (4.3) montre un exemple de tournée effectuée par le véhicule : il passe par les stations $Depot = 0, 1, 2, 3, 4, 5, 6 = Depot$, tout en faisant le plein d'hydrogène entre la station 1 et la station 2,

et ensuite entre la station 3 et la station 4.

D'autre part, on suppose que la micro-usine produit de l'hydrogène in situ à partir de l'eau par une combinaison de photolyse et d'électrolyse. L'hydrogène résultant est stocké dans un réservoir directement lié à la micro-usine, dont la capacité (en unités d'énergie) est désignée par C^{Tank} .

On suppose que l'espace temps $\{0, \dots, TMax\}$ est divisé en périodes $P_i = [p \times i, p \times (i + 1)[$, $i = 0, \dots, N - 1$, toutes d'une même longueur égale à p telle que $TMax = N \times p$. Par souci de simplicité, nous désignons une période P_i par son indice i . La figure (4.4) illustre 4 périodes qui valent chacune 2 unités de temps c'est-à-dire $p = 2$, on a les périodes suivantes :

- $P_0 = [2 \times 0, 2 \times (0 + 1)[= [0, 2[;$
- $P_1 = [2 \times 1, 2 \times (1 + 1)[= [2, 4[;$
- $P_2 = [2 \times 2, 2 \times (2 + 1) = [4, 6[;$
- $P_3 = [2 \times 3, 2 \times (3 + 1) = [6, 8[.$

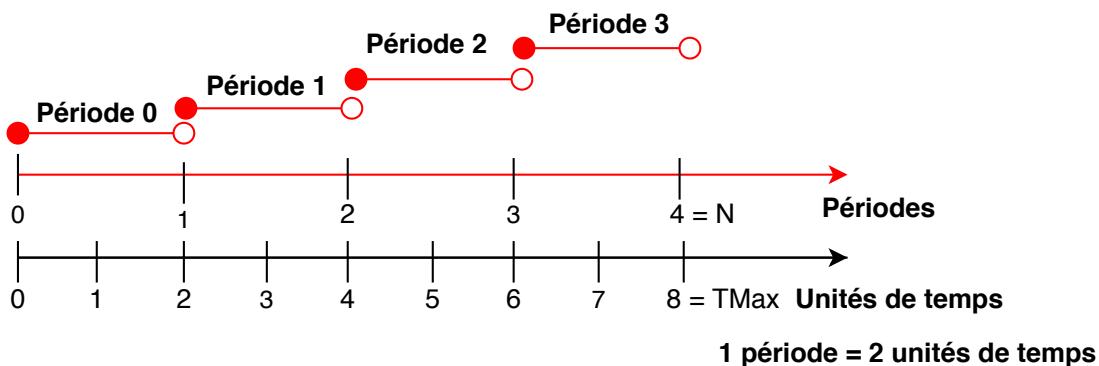


FIGURE 4.4 – Intervalle de temps correspondant à une période.

Si la micro-usine est active à un moment donné pendant la période i , alors elle est active pendant toute la période i , et produit R_i unités d'hydrogène avec R_i dépendant de la période i . A la date 0, la charge actuelle du réservoir de la micro-usine est égale à $H_0 \leq C^{Tank}$ et la micro-usine n'est pas active. Nous imposons que la même situation se produise à la date $TMax$.

La figure (4.5) présente un exemple de **stratégie de production** réalisée par la micro-usine : les périodes soulignées en rouge correspondent aux périodes où la micro-usine est activée. Les périodes en bleu correspondent aux périodes où la micro-usine est active.

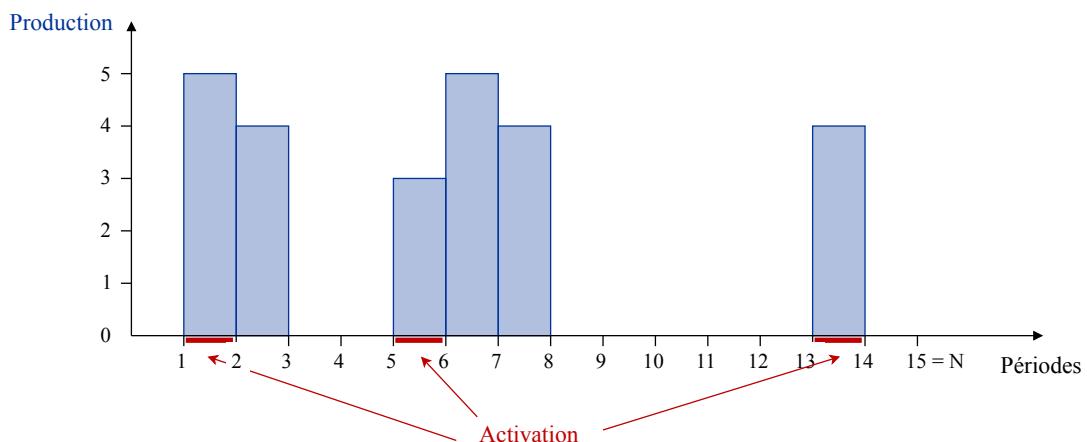


FIGURE 4.5 – Un exemple de stratégie de production.

Pour des raisons de sécurité, on suppose que le véhicule ne peut pas se recharger en hydrogène pendant que la micro-usine produit. Toute opération de recharge d'un véhicule doit commencer au début d'une période $i = 0, \dots, N - 1$ et se terminer à la fin de la période i . Etant donné que la recharge du véhicule et la production d'hydrogène s'excluent mutuellement, le véhicule peut attendre dans la micro-usine avant d'être autorisé à se recharger.

La production d'hydrogène a un coût, qui est décomposé ici en 2 composantes :

- Un coût d'activation, fixe et noté $Cost^F$, qui est facturé à chaque fois que la micro-usine est activée ;
- Un coût de production dépendant du temps, qui correspond à l'énergie produite pendant la période i , à condition que la micro-usine soit active pendant cette période : ce coût $Cost_i^V$ est indépendant de la quantité d'hydrogène réellement produite pendant la période i , et reflète les prix indexés sur le temps facturés par le fournisseur d'électricité.

La figure (4.6) affiche les coûts d'activation et les coûts de production en fonction du temps liés à la micro-usine de la figure (4.5).

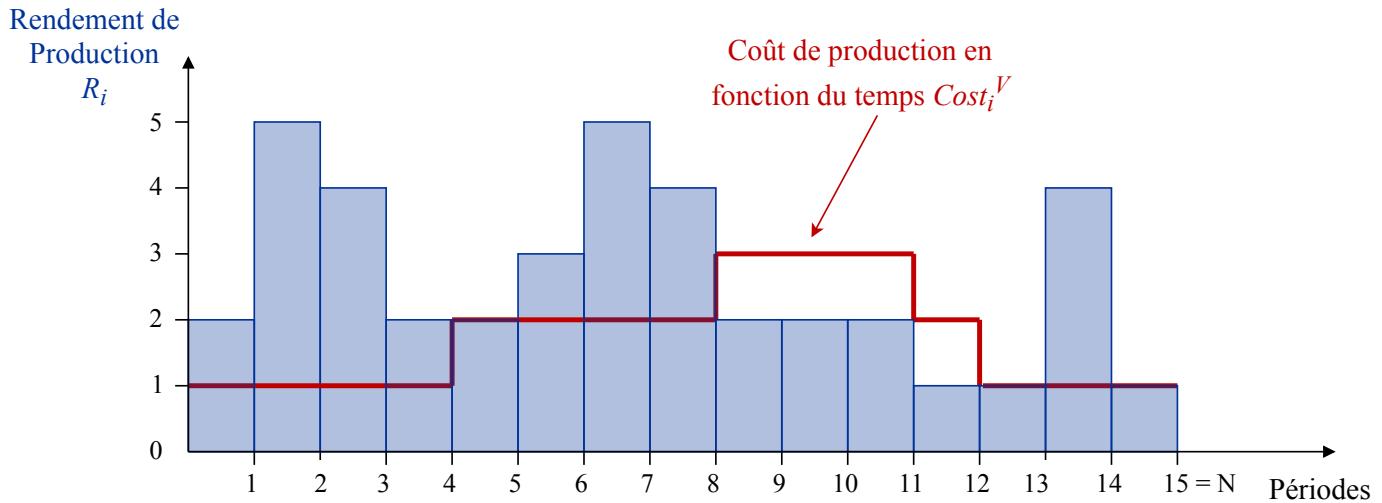


FIGURE 4.6 – Coûts de production et rendement en fonction du temps pour la micro-usine de la figure (4.5).

Notre problème **SMEPC** consiste à planifier à la fois les recharges du véhicule et la production de la micro-usine de manière à ce que :

- Le véhicule part du dépôt initial $Depot = 0$, visite toutes les stations $j = 1, \dots, M$ et revient au dépôt final $Depot = M + 1$ à une date $T \in [0, TMax]$, tout en se déplaçant, à chaque fois qu'il est nécessaire, vers la micro-usine afin de faire le plein d'hydrogène ;
- La micro-usine produit et stocke l'hydrogène nécessaire au véhicule de sorte que la quantité nécessaire à la recharge soit disponible à l'usine quand le véhicule se recharge ;
- Le coût $Cost$ de production d'hydrogène et le temps T de durée du parcours sont les plus faibles possibles. Nous fusionnons les deux coûts ci-dessus en un seul : $Cost + \alpha \times T$, où α est un coefficient d'échelle qui permet d'uniformiser les différentes composantes de la fonction objectif. Dans notre cas, on convertit le temps en coût.

Exemple 1 La figure (4.7) montre une solution du problème de synchronisation. On y voit les recharges du véhicule et la production de la micro-usine associée aux figures (4.3), (4.5), (4.6) dans le cas où $p = 2$,

$E_0 = 8$, $H_0 = 4$, $TMax = 30$, $Cost^F = 7$, $C^{Tank} = 15$, $C^{Veh} = 15$, $\alpha = 1$. Dans ce cas, le véhicule se recharge deux fois : la première recharge a lieu à la période 4, impliquant 14 unités d'hydrogène, et la deuxième recharge a lieu à la période 12, impliquant 11 unités d'hydrogène. Le temps T est égal à 30. Le coût d'activation global qui en résulte est de $3 \times 7 = 21$. Le coût de production dépendant du temps est égal à $1+1+2+2+2+1=9$. Ainsi, le coût global est égal à $21 + 9 + 30 = 60$.

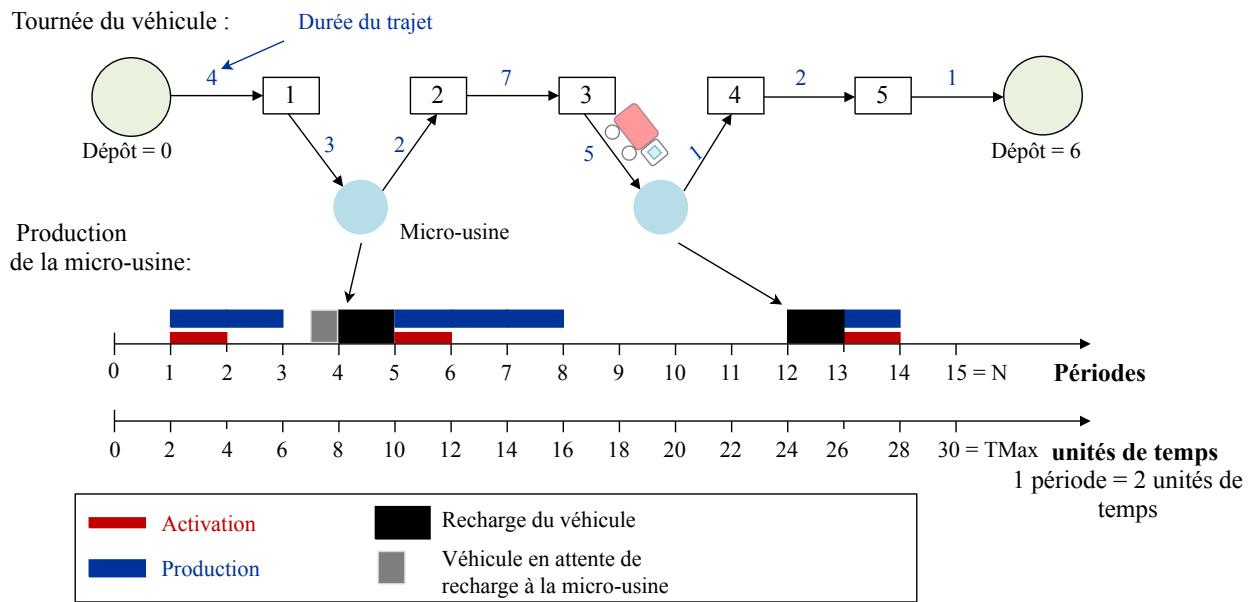


FIGURE 4.7 – Une solution possible de l'instance de SMEPC associée à la figure (4.3), à la figure (4.5) et à la figure (4.6).

Le tableau (4.1) résume toutes les entrées du problème SMEPC.

Noms	Significations
M	Nombre de stations (Dépôt et micro-usine exclus)
$\Gamma = (Depot = 0, 1, \dots, M, Depot = M + 1)$	Tournée fixe du véhicule (sans les recharges)
$TMax$	Le délai maximal pour que le véhicule puisse effectuer sa tournée
C^{Veh}	Capacité du réservoir en hydrogène du véhicule
E_0	Charge initiale en hydrogène du véhicule
Pour $j = 0, \dots, M, t_j$	Temps nécessaire pour aller de la station j à la station $j + 1$
Pour $j = 0, \dots, M, d_j$	Temps nécessaire pour aller de la station j à la micro-usine
Pour $j = 0, \dots, M, d_j^*$	Temps nécessaire pour aller de la micro-usine à la station j
Pour $j = 0, \dots, M, e_j$	Energie nécessaire pour aller de la station j à la station $j + 1$
Pour $j = 0, \dots, M, \varepsilon_j$	Energie nécessaire pour aller de la station j à la micro-usine
Pour $j = 0, \dots, M, \varepsilon_j^*$	Energie nécessaire pour aller de la micro-usine à la station j
C^{Tank}	Capacité de la citerne d'hydrogène
N	Nombre de périodes de production
p	Durée en unités de temps d'une période de production
H_0	Charge initiale de la citerne d'hydrogène
$Cost^F$	Coût d'activation de la micro-usine
Pour $i = 0, \dots, N - 1, P_i = [p \times i, p \times (i + 1)[$	Intervalle de temps correspondant à une période de production
Pour $i = 0, \dots, N - 1, R_i$	Rendement de production lié à la période i
Pour $i = 0, \dots, N - 1, Cost_i^V$	Coût de production lié à la période i

TABLE 4.1 – Entrées du problème SMEPC.

En analysant le problème SMEPC, on remarque qu'il peut être divisé en deux sous-problèmes illustrés à la figure (4.8) :

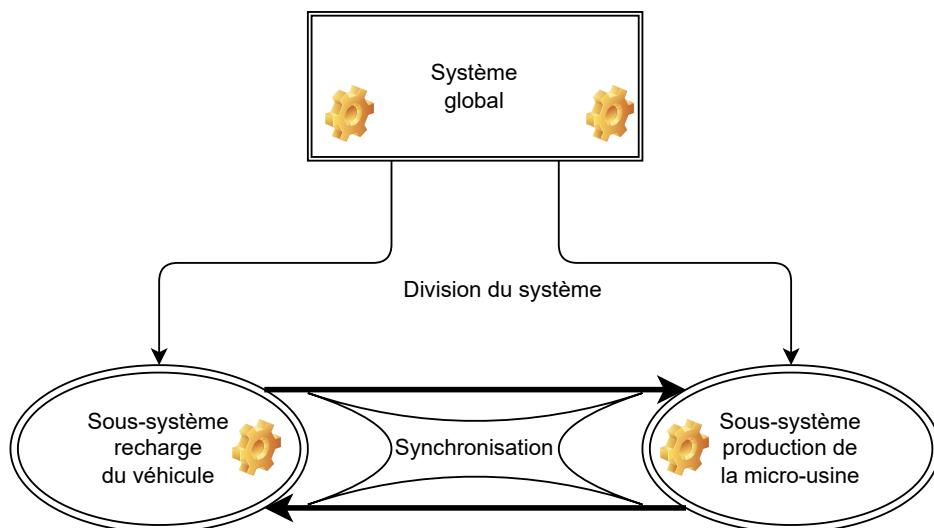


FIGURE 4.8 – Décomposition du problème SMEPC.

- Le problème *Vehicle-Driver (VD)* qui consiste à faire abstraction dans le problème **SMEPC** de la partie planification de la production d'hydrogène en supposant qu'on a une quantité infinie d'hydrogène à la micro-usine ;
- Le problème *Production-Manager (PM)* qui consiste à faire abstraction dans le problème **SMEPC** de la partie planification des recharges en hydrogène du véhicule en supposant qu'on connaît toutes les demandes du véhicule.

Nous venons de présenter le modèle **SMEPC**. La section suivante sera consacrée à la présentation de la formulation mathématique de ce problème.

4.3 Formulation mathématique

Dans cette partie, une formulation du problème SMEPC est proposée dans un premier temps puis, celle-ci est linéarisée pour finalement obtenir un Programme Linéaire Mixte (MILP).

4.3.1 Variables

1. Les variables du problème **PM** sont :

$z = (z_i, i = 0, \dots, N - 1), z_i \in \{0, 1\}$

$$z_i = \begin{cases} 1 & \text{si la micro-usine est active pendant la période } i \\ 0 & \text{sinon.} \end{cases}$$

$y = (y_i, i = 0, \dots, N - 1), y_i \in \{0, 1\}$:

$$y_i = \begin{cases} 1 & \text{si la micro-usine est activée au début de la période } i \\ 0 & \text{sinon.} \end{cases}$$

$V^{Tank} = (V_i^{Tank}, i = 0, \dots, N)$, V_i^{Tank} est une valeur non négative qui représente la quantité d'hydrogène dans la citerne au début de la période i . Nous tenons compte ici d'une période fictive N afin d'exprimer le fait que la quantité d'hydrogène dans la citerne de la micro-usine à la fin du processus devrait être au moins égale H_0 ;

$\delta = (\delta_i, i = 0, \dots, N - 1), \delta_i \in \{0, 1\}$:

$$\delta_i = \begin{cases} 1 & \text{si le véhicule se recharge durant la période } i \\ 0 & \text{sinon.} \end{cases}$$

$L^* = (L_i^*, i = 0, \dots, N - 1)$, L_i^* est une valeur non négative. Si $\delta_i = 1$, L_i^* est la quantité d'hydrogène donnée par l'usine au véhicule pendant la période i .

2. Les variables du problème **VD** sont :

$x = (x_j, j = 0, \dots, M), x_j \in \{0, 1\}$:

$$x_j = \begin{cases} 1 & \text{si le véhicule se recharge en hydrogène lorsqu'il se déplace de la station } j \text{ à la station } j + 1 \\ 0 & \text{sinon.} \end{cases}$$

$L = (L_j, j = 0, \dots, M)$, L_j est une valeur non négative qui représente la quantité d'hydrogène donnée par l'usine au véhicule lorsqu'il se déplace de la station j à la station $j + 1$;

- $T = (T_j, j = 0, \dots, M + 1)$, T_j est une valeur non négative qui représente la date d'arrivée du véhicule à la station j ;
- $T^* = (T_j^*, j = 0, \dots, M + 1)$, T_j^* est une valeur non négative. Si $x_j = 1$, T_j^* est la date à laquelle le véhicule commence à se recharger en hydrogène entre la station j et la station $j + 1$;
- $V^{Veh} = (V_j^{Veh}, j = 0, \dots, M + 1)$ V_j^{Veh} est une valeur non négative qui représente la quantité d'hydrogène dans le réservoir du véhicule lorsqu'il arrive à la station j .

Ces variables seront contraintes comme suit :

4.3.2 Fonction objectif

La fonction objectif concerne à la fois la minimisation du coût de production d'hydrogène et de la durée de la tournée. On convertit le temps en coût à l'aide d'un coefficient qu'on nomme α . On passe donc du bi-objectif à un objectif unique de la façon suivante :

$$\text{Min} \left\{ \sum_{i=0}^{N-1} [(Cost^F \times y_i) + (Cost_i^V \times z_i)] + \alpha \times T_{M+1} \right\} \quad (4.1)$$

Où α est le facteur de conversion du temps en coût économique.

4.3.3 Contraintes du problème PM

Les contraintes du problème PM sont :

$$z_0 = y_0, \delta_0 = 0 \quad (4.2a)$$

$$z_i + \delta_i \leq 1 \quad \forall i = 0, \dots, N - 1 \quad (4.2b)$$

$$y_i = 1 \rightarrow (z_{i-1} = 0 \wedge z_i = 1) \quad \forall i = 1, \dots, N - 1 \quad (4.2c)$$

$$V_0^{Tank} = H_0 \quad (4.3a)$$

$$V_N^{Tank} \geq H_0 \quad (4.3b)$$

$$V_i^{Tank} \leq C^{Tank} \quad \forall i = 0, \dots, N - 1 \quad (4.3c)$$

$$V_{i+1}^{Tank} = V_i^{Tank} + z_i \times R_i - L_i^* \quad \forall i = 0, \dots, N - 1 \quad (4.3d)$$

$$L_i^* \geq 0 \quad \forall i = 0, \dots, N - 1 \quad (4.3e)$$

$$L_i^* \leq C^{Tank} \times \delta_i \quad \forall i = 0, \dots, N - 1 \quad (4.3f)$$

La contrainte (4.2a) définit l'état initial de l'usine. La contrainte (4.2b) traduit le fait que la production d'hydrogène et la recharge du véhicule ne se font pas simultanément. La contrainte (4.2c) traduit le fait que démarrer la micro-usine à la période i signifie que la micro-usine ne produisait pas de l'hydrogène à la période $i - 1$ et commence à produire l'hydrogène à la période i .

La contrainte (4.3a) signifie que la quantité d'hydrogène dans la citerne de la micro-usine au début de la tournée du véhicule est H_0 . La contrainte (4.3b) signifie que la quantité d'hydrogène dans

la citerne à la fin du processus doit au moins être égale à H_0 . La contrainte (4.3c) signifie que durant toute la tournée, la quantité d'hydrogène dans la citerne de la micro-usine ne dépasse pas la capacité maximal C^{Tank} de la citerne. La contrainte (4.3d) signifie que le stock d'hydrogène dans la citerne à la période i diminue de la quantité d'hydrogène donnée au véhicule pendant la période i (s'il y a recharge) et augmente de la quantité d'hydrogène produite à la période i (s'il y a production).

4.3.4 Contraintes du problème VD

Les contraintes du problème VD sont :

$$V_0^{Veh} = E_0 \quad (4.4a)$$

$$V_{M+1}^{Veh} \geq E_0 \quad (4.4b)$$

$$V_j^{Veh} \leq C^{Veh} \quad \forall j = 1, \dots, M + 1 \quad (4.4c)$$

$$V_j^{Veh} \geq \varepsilon_j \quad \forall j = 0, \dots, M \quad (4.4d)$$

$$V_{j+1}^{Veh} = V_j^{Veh} - e_j + x_j \times (e_j - \varepsilon_j - \varepsilon_{j+1}^*) + L_j \quad \forall j = 0, \dots, M \quad (4.4e)$$

$$L_j \geq 0 \quad \forall j = 0, \dots, M \quad (4.4f)$$

$$L_j \leq C^{Veh} \times x_j \quad \forall j = 0, \dots, M \quad (4.4g)$$

$$L_j \leq C^{Veh} + \varepsilon_j - V_j^{Veh} \quad \forall j = 0, \dots, M \quad (4.4h)$$

$$T_0 = 0 \quad (4.5a)$$

$$T_{M+1} \leq TMax \quad (4.5b)$$

$$T_{j+1} \geq (1 - x_j) \times (T_j + t_j) + x_j \times (T_j^* + p + d_{j+1}^*) \quad \forall j = 0, \dots, M \quad (4.5c)$$

$$T_j^* \geq T_j + d_j \quad \forall j = 0, \dots, M \quad (4.5d)$$

La contrainte (4.4a) signifie que lorsque le véhicule commence sa tournée au dépôt initial $Depot = 0$, la quantité d'hydrogène dans son réservoir est E_0 . Cette valeur est une donnée. La contrainte (4.4b) signifie que le véhicule arrive au dépôt final $Depot = M + 1$ avec au moins la quantité d'hydrogène E_0 qu'il avait au dépôt initial $Depot = 0$. La contrainte (4.4c) signifie que durant toute la tournée, la quantité d'hydrogène dans le réservoir du véhicule ne dépasse pas sa capacité maximal C^{Veh} car le réservoir d'hydrogène a une capacité limitée. La contrainte (4.4d) signifie qu'à tout moment après une station, le véhicule doit pouvoir se rendre à la micro-usine et faire le plein, et s'appuie sur l'inégalité triangulaire pour les coefficients énergétiques e_j et ε_j . La contrainte (4.4e) signifie que lorsque le véhicule est à la station j , s'il décide de continuer sa tournée en allant à la station $j + 1$ alors la quantité d'hydrogène dans son réservoir diminue de e_j . De plus, elle signifie aussi que lorsque le véhicule est à la station j , s'il décide de partir se recharger alors la quantité d'hydrogène dans son réservoir diminue du détour induit $\varepsilon_j + \varepsilon_{j+1}^*$ et augmente de la quantité d'hydrogène rechargée. Les contraintes (4.4f), (4.4g) et (4.4h) signifient que la quantité d'hydrogène L_j rechargée par le véhicule ne doit jamais être plus grande que la capacité maximal du véhicule et la différence entre C^{Veh} et la recharge courante du réservoir du véhicule. Nous devons néanmoins veiller à éviter de produire plus que nécessaire.

La contrainte (4.5a) signifie que le véhicule commence sa tournée au dépôt initial $Depot = 0$ à

la date 0. La contrainte (4.5b) signifie le véhicule arrive au dépôt final $Depot = M + 1$ au plus tard à la date $TMax$. La contrainte (4.5c) signifie que lorsque le véhicule est à la station j , s'il décide de continuer sa tournée en allant à la station $j + 1$ alors sa date d'arrivée T_j à la station j augmente de t_j . Alors que, si le véhicule décide de partir se recharger alors sa date d'arrivée à la station $j + 1$ est $T_j^* + p + d_{j+1}^*$. La contrainte (4.5d) signifie que le véhicule peut commencer sa recharge lorsqu'il est arrivé à la micro-usine.

4.3.5 Contraintes de synchronisation

Pour obtenir une formulation complète, nous devons expliquer la façon dont les activités du véhicule et de la micro-usine sont synchronisées. Pour cela, il faut introduire une variable de synchronisation $U = (U_{i,j}, i = 0, \dots, N - 1, j = 0, \dots, M)$ prenant des valeurs booléennes, qui nous indiqueront, dans le cas où le véhicule décide de faire le plein entre la station j et la station $j + 1$, durant quelle période i il le fera.

$$U_{i,j} = \begin{cases} 1 & \text{si le véhicule se recharge en hydrogène durant la période } i \text{ lors de sa tournée de } j \text{ à } j + 1. \\ 0 & \text{sinon.} \end{cases}$$

Nous complétons notre formulation **SMEPC** avec les contraintes de synchronisation suivantes :

$$\sum_{i=0, \dots, N-1} U_{i,j} = x_j \quad \forall j = 0, \dots, M \quad (4.6a)$$

$$\delta_i = \sum_{j=0, \dots, M} U_{i,j} \quad \forall i = 0, \dots, N-1 \quad (4.6b)$$

$$T_j^* \geq \sum_{i=0, \dots, N-1} p \times i \times U_{i,j} \quad \forall j = 0, \dots, M \quad (4.6c)$$

$$x_j = 1 \rightarrow \sum_{i=0, \dots, N-1} p \times i \times U_{i,j} \geq T_j + d_j \quad \forall j = 0, \dots, M \quad (4.6d)$$

$$L_i^* \geq \sum_{j=0, \dots, M} U_{i,j} \times L_j \quad \forall i = 0, \dots, N-1 \quad (4.6e)$$

Les contraintes (4.6) signifient que le véhicule se recharge en hydrogène durant une unique période $i = 0, \dots, N - 1$ lors de sa tournée de la station j à la station $j + 1$. De plus, la quantité rechargée est au moins L_j .

4.4 Formulation par programmation linéaire en variables mixtes : $MILP_{SMEPC}$

Dans le paragraphe précédent, on a associé au problème SMEPC une formulation mathématique avec certaines contraintes linéaires et d'autres plutôt logiques. Ici, on linéarise ces dernières en nous servant du Big M lorsque c'est nécessaire. A la suite de quoi on obtiendra un MILP (Mixed-Integer Program). Pour obtenir le MILP du problème **SMEPC**, il suffit de linéariser d'abord les contraintes contenant une implication (4.2c) et (4.6d). Ensuite de linéariser les contraintes quadratiques (4.5c) et (4.6e).

Le tableau (4.2) regroupe la linéarisation des contraintes du modèle. La variable m est une variable de linéarisation du modèle. On appellera ce Programme Linéaire Mixte : $MILP_{SMEPC}$.

Numéro	Contraintes	Bornes	Linéarisation
4.2a	$z_0 - y_0 = 0, \delta_0 = 0$		
4.2b	$z_i + \delta_i \leq 1$	$\forall i = 0, \dots, N-1$	
4.2c	$y_i = 1 \rightarrow (z_{i-1} = 0 \wedge z_i = 1)$	$\forall i = 1, \dots, N-1$	$y_i - z_i \leq 0$ $y_i + z_{i-1} \leq 1$ $z_i - z_{i-1} - y_i \leq 0$
4.3a	$V_0^{Tank} = H_0$		
4.3b	$V_N^{Tank} \geq H_0$		
4.3c	$V_i^{Tank} \leq C^{Tank}$	$\forall i = 0, \dots, N-1$	
4.3d	$V_{i+1}^{Tank} = V_i^{Tank} + z_i \times R_i - L_i^*$	$\forall i = 0, \dots, N-1$	
4.3e	$L_i^* \geq 0$	$\forall i = 0, \dots, N-1$	
4.3f	$L_i^* \leq C^{Tank} \times \delta_i$	$\forall i = 0, \dots, N-1$	
4.4a	$V_0^{Veh} = E_0$		
4.4b	$V_{M+1}^{Veh} \geq E_0$		
4.4c	$V_j^{Veh} \leq C^{Veh}$	$\forall j = 1, \dots, M+1$	
4.4d	$V_j^{Veh} \geq \varepsilon_j$	$\forall j = 0, \dots, M$	
4.4e	$V_{j+1}^{Veh} = V_j^{Veh} - e_j + (e_j - \varepsilon_j - \varepsilon_{j+1}^*) \times x_j + L_j$	$\forall j = 0, \dots, M$	
4.4f	$L_j \geq 0$	$\forall j = 0, \dots, M$	
4.4g	$L_j \leq C^{Veh} \times x_j$	$\forall j = 0, \dots, M$	
4.4h	$L_j \leq C^{Veh} + \varepsilon_j - V_j^{Veh}$	$\forall j = 0, \dots, M$	
4.5a	$T_0 = 0$		
4.5b	$T_{M+1} \leq TMax$		
4.5c	$T_{j+1} \geq (1 - x_j) \times (T_j + t_j) + x_j \times (T_j^* + p + d_{j+1}^*)$	$\forall j = 0, \dots, M$	$T_{j+1} - T_j - t_j \geq -2 \times TMax \times x_j$ $T_{j+1} - (T_j^* + p + d_{j+1}^*) \geq -2 \times TMax \times (1 - x_j)$
4.5d	$T_j^* \geq T_j + d_j$	$\forall j = 0, \dots, M$	
4.6a	$\sum_{i=0, \dots, N-1} U_{i,j} = x_j$	$\forall j = 0, \dots, M$	
4.6b	$\delta_i = \sum_{j=0, \dots, M} U_{i,j}$	$\forall i = 0, \dots, N-1$	
4.6c	$T_j^* \geq \sum_{i=0, \dots, N-1} p \times i \times U_{i,j}$	$\forall j = 0, \dots, M$	
4.6d	$x_j = 1 \rightarrow \sum_{i=0, \dots, N-1} p \times i \times U_{i,j} \geq T_j + d_j$	$\forall j = 0, \dots, M$	$\sum_{i=0, \dots, N-1} p \times i \times U_{i,j} \geq T_j + d_j - 2 \times TMax \times (1 - x_j)$
4.6e	$L_i^* = \sum_{j=0, \dots, M} U_{i,j} \times L_j$	$\forall i = 0, \dots, N-1$	$L_j \leq L_i^* + (C^{Veh} + C^{Tank}) \times (1 - U_{i,j})$ $L_i^* \leq L_j + (C^{Veh} + C^{Tank}) \times (1 - U_{i,j})$

TABLE 4.2 – Linéarisation des contraintes.

Théorème 2 $MILP_{SMEPC}$ a une solution réalisable si et seulement si $SMEPC$ a une solution de même coût.

La démonstration de ce théorème se trouve en annexe à la section 4.9.1.

On vient de modéliser le problème **SMEPC** en présentant ses variables, sa fonction objectif et ses contraintes. La partie qui suit est consacrée à la présentation de la relaxation linéaire de la formulation $MILP_{SMEPC}$.

4.5 Relaxation linéaire de la formulation $MILP_{SMEPC}$ et contraintes additionnelles STC et EC : $RMILP_{SMEPC}$

On désigne par $RMILP_{SMEPC}$ la relaxation linéaire de $MILP_{SMEPC}$, et par \bar{Z}_r sa valeur optimale. Pour obtenir le modèle fractionnaire correspondant au modèle linéaire présenté, il suffit de

rendre toutes les variables du modèle linéaire réelles. Les variables deviendront :

- $z = (z_i, i = 0, \dots, N - 1), z_i \in [0, 1];$
- $y = (y_i, i = 0, \dots, N - 1), y_i \in [0, 1];$
- $V^{Tank} = (V_i^{Tank}, i = 0, \dots, N), V^{Tank} \in \mathbb{R}^+;$
- $\delta = (\delta_i, i = 0, \dots, N - 1), \delta_i \in [0, 1];$
- $L^* = (L_i^*, i = 0, \dots, N - 1), L^* \in \mathbb{R}^+;$
- $x = (x_j, j = 0, \dots, M), x_j \in [0, 1];$
- $L = (L_j, j = 0, \dots, M), L_j \in \mathbb{R}^+;$
- $T = (T_j, j = 0, \dots, M + 1), T_j \in \mathbb{R}^+;$
- $T^* = (T_j^*, j = 0, \dots, M + 1), T_j^* \in \mathbb{R}^+;$
- $V^{Veh} = (V_j^{Veh}, j = 0, \dots, M + 1) V_j^{Veh} \in \mathbb{R}^+$
- $U_{i,j} \in [0, 1]$
- $m_{i,j} \in [0, 1].$

Souvent, les techniques de BigM induisent des relaxations linéaires très faibles. Mais ici, nous avons ce qui suit.

Lemme 1 Si $RMILP_{SMEPC}$ est réalisable alors $\bar{Z}_r = 0$.

Pour faire décoller la relaxation linéaire de zéro on a remplacé les contraintes (4.6e) par les contraintes suivantes :

$$L_i^* = \sum_{j=0, \dots, M} m_{i,j} \quad (4.7a)$$

$$m_{i,j} \geq 0 \quad (4.7b)$$

$$m_{i,j} \leq C^{Veh} \times U_{i,j} \quad (4.7c)$$

$$m_{i,j} \leq L_j \quad (4.7d)$$

$$m_{i,j} \geq L_j - C^{Veh} \times (1 - U_{i,j}) \quad (4.7e)$$

Lemme 2 Avec les contraintes précédentes, si $RMILP_{SMEPC}$ est réalisable alors $\bar{Z}_r > 0$.

La démonstration de ce lemme se trouve en annexe à la section 4.9.2.

4.5.1 Contraintes additionnelles EC et STC

Plusieurs contraintes peuvent être ajoutées pour renforcer la relaxation linéaire. Pour atteindre cet objectif nous introduisons les données suivantes.

Pour tout $j = 1, \dots, M$, nous fixons :

$D_j = \sum_{k=0, \dots, j-1} t_k + d_j$: la date d'arrivée la plus proche à la micro-usine pour une première recharge après la station j ;

$D_j^* = d_{j+1}^* + \sum_{k=j+1, \dots, M} t_k$: la date la plus tardive pour terminer le trajet après avoir fait le plein à la station j ;

$\tau_m(j) = \lceil \frac{D_j}{p} \rceil$: la période la plus proche d'une éventuelle recharge à la station j ;

$\tau_M(j) = N - 1 - \lceil \frac{D_j^*}{p} \rceil$: la période la plus tardive d'une recharge possible à la station j .

Contraintes STC : Simple Time Constraints

$$T_{j+1}^* \geq T_j^* \quad \forall j = 0, \dots, M \quad (4.8a)$$

$$T_{j+1} \geq T_j + t_j + x_j \times (d_j + d_{j+1}^* - t_j) \quad \forall j = 0, \dots, M \quad (4.8b)$$

Les inégalités (4.8a) et (4.8b) sont directement considérées comme valides pour MILP_{SMEPC}. (4.8a) et (4.8b) assurent que les temps forment des séquences non décroissantes.

Contraintes EC : Energy Constraints

$$L_j \geq x_j \times (\varepsilon_j + \varepsilon_{j+1}^* - e_j) \quad \forall j = 0, \dots, M \quad (4.9)$$

En considérant (4.9), le véhicule quittant une station quelconque arrivera à la suivante avec plus d'hydrogène après une recharge que s'il suivait la route directe entre les deux stations.

$$U_{i,j} = 0, i < \tau_m(j) \text{ ou } i > \tau_M(j) \quad \forall j = 0, \dots, M \quad (4.10)$$

Les inégalités (4.10) reflètent simplement les définitions de $\tau_m(j)$ et $\tau_M(j)$, pour tout j .

$$\sum_{i=0, \dots, \tau_M(j)} L_i^* \geq \sum_{k=0, \dots, j} L_k \quad \forall j = 0, \dots, M \quad (4.11)$$

(4.11) exprime le fait que la quantité totale rechargée par le véhicule à la station j ne dépasse pas la quantité totale d'hydrogène fournie par la micro-usine au véhicule jusqu'à $\tau_M(j)$. (4.11) a les conséquences suivantes :

Soit $F_{j+1} = F_j + e_j + x_j \times (\varepsilon_j + \varepsilon_{j+1}^* - e_j)$ avec $F_0 = 0$. F_j est l'énergie utilisée par le véhicule de 0 à j .

D'après (4.3d) et (4.4e), on a les contraintes (4.12a) et (4.12b).

$$V_{j+1}^{Veh} = E_0 - F_{j+1} + \sum_{k=0, \dots, j} L_k \quad \forall j = 0, \dots, M \quad (4.12a)$$

$$V_{\tau_M(j)+1}^{Tank} = H_0 + \sum_{i=0, \dots, \tau_M(j)} R_i \times z_i - \sum_{i=0, \dots, \tau_M(j)} L_i^* \quad \forall j = 0, \dots, M \quad (4.12b)$$

Donc, V_{j+1}^{Veh} et $V_{\tau_M(j)+1}^{Tank}$ étant des valeurs positives, on a les contraintes (4.13a) et (4.13b).

$$E_0 + \sum_{k=0, \dots, j} L_k \geq F_{j+1} \quad \forall j = 0, \dots, M \quad (4.13a)$$

$$H_0 + \sum_{i=0, \dots, \tau_M(j)} R_i \times z_i \geq \sum_{i=0, \dots, \tau_M(j)} L_i^* \quad \forall j = 0, \dots, M \quad (4.13b)$$

On obtient donc :

EC1 : ce sont les contraintes (4.14a) et (4.14b)

$$\sum_{i=1,\dots,\tau_M(j)} R_i \times z_i \geq F_{j+1} - E_0 - H_0 \quad \forall j = 0, \dots, M \quad (4.14a)$$

$$\sum_{i=1,\dots,N-1} R_i \times z_i \geq F_{M+1} \quad (4.14b)$$

La contrainte (4.14a) est déduite des contraintes (4.13b), (4.11) et (4.13a). La contrainte (4.14b) exprime le fait que la quantité d'hydrogène produite par la micro-usine doit être suffisante pour réaliser toute la tournée. La variable binaire y_i est égale à 1 lorsque la micro-usine est activée à la période i . Ainsi, la valeur $\sum_{i=0,\dots,\tau_M(j)} y_i$ indique le nombre d'intervalle de production entre les périodes 0 et $\tau_M(j)$. Pendant chacun de ces intervalles, la production d'hydrogène ne peut pas dépasser la capacité de la micro-usine. Par conséquent on a la contrainte (4.15).

$$C^{Tank} \times \sum_{i=0,\dots,\tau_M(j)} y_i \geq \sum_{i=0,\dots,\tau_M(j)} R_i \times z_i \quad j = 0, \dots, M \quad (4.15)$$

D'après (4.14a), cela implique qu'on a les contraintes (4.16a) et (4.16b).

EC2 :

$$C^{Tank} \times \sum_{i=0,\dots,\tau_M(j)} y_i \geq F_j - E_0 - H_0 \quad \forall j = 0, \dots, M \quad (4.16a)$$

$$C^{Tank} \times \sum_{i=0,\dots,N-1} y_i \geq F_{M+1} \quad (4.16b)$$

De la même manière, à partir de (4.3f), on a la contrainte (4.17).

$$\sum_{i=0,\dots,\tau_M(j)} C^{Tank} \times \delta_i \geq \sum_{i=0,\dots,\tau_M(j)} L_i^* \quad \forall j = 0, \dots, M : \quad (4.17)$$

Nous obtenons donc les contraintes (4.18a) et (4.18b).

EC2' :

$$C^{Tank} \times \sum_{i=0,\dots,\tau_M(j)} \delta_i \geq F_j - E_0 \quad \forall j = 0, \dots, M \quad (4.18a)$$

$$C^{Tank} \times \sum_{i=0,\dots,N-1} \delta_i \geq F_{M+1} \quad (4.18b)$$

La contrainte (4.18a) est déduite des contraintes (4.17), (4.11) et (4.13a). De façon similaire à (4.11), on peut voir qu'on a la contrainte (4.19).

$$\sum_{i=\tau_m(j),\dots,N-1} L_i^* \geq \sum_{k=j,\dots,M} L_k \quad \forall j = 0, \dots, M \quad (4.19)$$

(4.19) exprime le fait que la quantité totale rechargée par le véhicule à partir de la station j ne dépasse pas la quantité totale d'hydrogène fournie par la micro-usine à partir de $\tau_m(j)$. Elle peut également être utilisée pour générer des inégalités similaires à celles de type (4.14a) et (4.16a).

Enfin, nous proposons quelques inégalités. Soient μ_j^0 et μ_j^* dont la signification est :

- $\mu_j^0 = \sum_{k=0, \dots, j-1} e_k + \varepsilon_j$, pour tout $j = 1, \dots, M$. μ_j^0 est la consommation d'énergie du véhicule qui commence au dépôt et finit à la micro-usine avant de faire le plein à la station j .
- $\mu_j^* = \varepsilon_{j+1}^* + \sum_{k=j+1, \dots, M} e_k$ pour tout $j = 0, \dots, M$. μ_j^* est la consommation d'énergie du véhicule partant de la micro-usine après une recharge en j et finissant au dépôt.

En fonction du fait que la consommation minimale du véhicule dépasse la quantité initiale d'hydrogène ou la capacité de son réservoir, les contraintes (4.20a) et (4.20b) peuvent être obtenues.

EC3 :

$$\sum_{k=0, \dots, j} x_k \geq 1 \quad \forall j = 0, \dots, M, \mu_j^0 > E_0 \quad (4.20a)$$

$$\sum_{k=j+1, \dots, M} x_k \geq 1, \quad \forall j = 0, \dots, M, \mu_j^* > C^{Veh} - E_0 \quad (4.20b)$$

Ces dernières contraintes expriment le fait que si $\mu_j^0 > E_0$ ou $\mu_j^* > C^{Veh} - E_0$ alors le véhicule devra se recharger au moins une fois pour pouvoir réaliser sa tournée.

4.6 Complexité du modèle SMEPC

Dans cette partie, nous montrons que, même lorsque le coût fixe de production $Cost^F$ est nul, le SMEPC se réduit au problème du sac à dos. Cela nous permet d'affirmer le théorème 3.

Théorème 3 *SMEPC est NP-difficile.*

La démonstration de ce théorème se trouve en annexe à la section 4.9.3.

Après la présentation des résultats des expérimentations, nous nous attaquerons à la résolution proprement dites du problème à l'aide d'un schéma de programmation dynamique.

4.7 Expérimentations numériques

4.7.1 Objectifs et contexte technique

L'objectif des expérimentations est de tester les formulations linéaires que nous avons proposé. Les modèles ont été implémentés en C++ et l'IDE utilisé est Visual Studio 2017. Les expérimentations sont réalisées sur un ordinateur équipé d'un processeur AMD EPYC 7H12 64-Core, 512 Go de RAM et fonctionnent sous Gnu/linux Ubuntu 20.04.2. Une première série d'expérimentations est réalisée au cours de laquelle CPLEX 12.10 est utilisé en mode *8 threads* et le temps maximum du CPU est fixé à 3 heures. Les bornes supérieures obtenues seront utilisées comme valeurs de référence pour calculer les gaps tout au long de ce document. Une deuxième série d'expérimentations est réalisée au cours de laquelle CPLEX 12.10 est utilisé en mode *single-threads* et le temps maximum du CPU est fixé à 1 heure.

4.7.2 Procédés de génération d'instances du SMEPC

Pour faire des expérimentations, on a besoin d'instances du problème SMEPC. Étant donné que ce dernier est un nouveau problème, on a dû concevoir des procédés de génération de ses instances.

On a un ensemble de 50 instances qui se subdivisent en deux paquets d'instances (paquet INST_CTE et paquet INST_VAR) car elles sont générées avec deux procédés différents. On a fixé que toutes nos entrées soient des valeurs entiers donc toutes nos variables le seront aussi. Dans cette section, on décrit les deux procédés de génération.

Le paquet d'instances INST_VAR a été généré avec une loi uniforme c'est pour cette raison que ces instances se « ressemblent ». Pour pouvoir tester les performances de nos modèles on avait besoin d'instances suffisamment « variées », on a donc généré les instances du paquet INST_CTE. La génération d'instances ici consiste à générer le nombre de stations, les coordonnées de ces stations, les coordonnées du dépôt et les coordonnées de la micro-usine, le temps maximal, la durée d'une période, les rendements de production, les couts variables , les couts fixes de production, les quantités d'hydrogènes initiales dans le véhicule et dans la citerne et les capacités du véhicule et de la citerne.

Certaines étapes de la génération sont communes aux deux méthodes à l'instar de :

- $xrectangle, yrectangle$: Limite des coordonnées des stations respectivement sur l'axe des abscisses et l'axe des ordonnées. On considère que deux stations ne peuvent pas avoir les mêmes coordonnées. On fixe le rectangle dans lequel on place les stations aléatoirement, une station est un point dans le plan. Les valeurs entières $xrectangle$ et $yrectangle$ représentent respectivement la longueur en abscisse et en ordonné de ce rectangle (Voir figure 4.9). Avec par exemple $(xrectangle, yrectangle) \in [1, M]^2$.

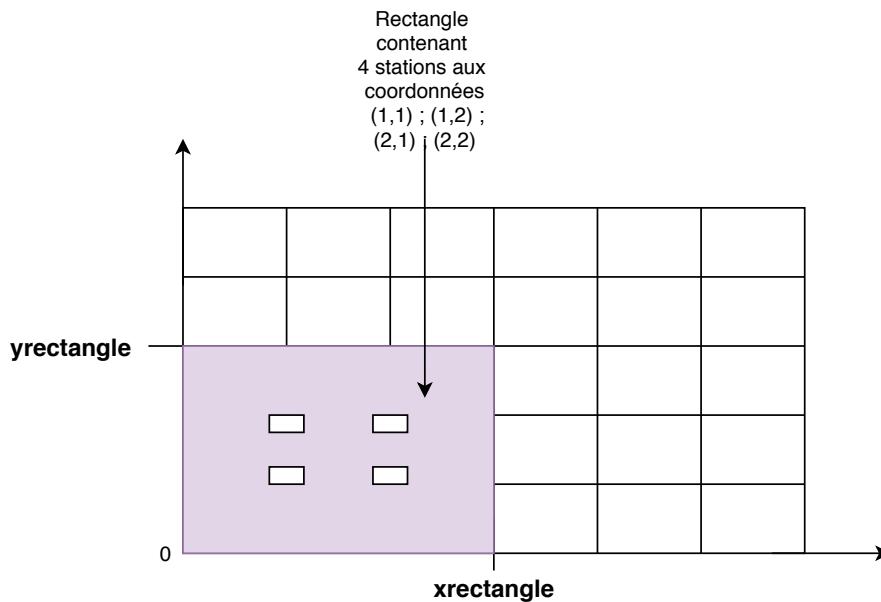


FIGURE 4.9 – Plan contenant les stations.

Si le dépôt a pour coordonnées (a, b) alors la micro-usine aura pour coordonnées $(a, b + 1)$ car on impose qu'il y ait une distance de 1 unité entre le dépôt et la micro-usine.

- A partir de ces coordonnées, la matrice des valeurs de distances est calculée comme la valeur arrondie de la distance euclidienne entre deux stations, et la matrice des valeurs d'énergie comme la valeur de la distance de Manhattan. Autrement dit, les valeurs de distances d_j, d_j^*, t_j et les valeurs d'énergies $e_j, \varepsilon_j, \varepsilon_j^*$ correspondent respectivement à l'arrondi (partie entière supérieure) de la distance euclidienne et à la distance de Manhattan.

Génération du paquet d'instances INST_VAR

On génère 30 instances avec le procédé de génération décrit par la suite. Nous fixons M et p , et générerons aléatoirement les stations j et la station particulière *Depot* comme point de l'espace \mathbb{R}^2 . On génère des instances aléatoirement de la façon suivante :

1. On fixe aléatoirement M le nombre de stations ($M > 2$). Car on doit avoir au moins deux stations pour réaliser les tâches de transport et de livraison.
2. On fixe aléatoirement p la taille d'une période.
3. On fixe le paramètre de la fonction objectif $\alpha = 1$.
4. On fixe E_0 , par exemple $E_0 \in [1, M]$. E_0 doit au moins être égale à 1 car il faut pouvoir se rendre à la micro-usine à partir du dépôt initial.
5. On fixe le paramètre entier *param_vmax*, par exemple *param_vmax* $\in [2, 3]$. On calcule C^{Veh} tel que $C^{Veh} = \frac{\sum_{j \leq M} e_j}{param_vmax} + E_0$. L'idée ici est d'imposer qu'il faut au moins *param_vmax* recharges pour pouvoir finir la tournée.
6. On fixe le paramètre entier *param_cit* $\in [1, 3]$. On calcule C^{Tank} tel que $C^{Tank} = param_cit \times C^{Veh}$. L'idée ici est d'imposer que si la citerne est pleine on peut remplir au plus *param_cit* fois le réservoir du véhicule sans production.
7. On fixe $Cost^F$, par exemple $Cost^F \in [1, 3 \times M]$.
8. On fixe H_0 , par exemple $H_0 \in [0, M]$.
9. On fixe le paramètre entier *param_tmax*, par exemple *param_tmax* $\in [2, 3]$. On calcule $TMax$ tel que $TMax \geq \sum_{j \leq M} d_j \times param_tmax$. On fixe $TMax$ ainsi pour être sûr d'avoir suffisamment de temps pour finir la tournée.
10. On calcule $N = \lceil \frac{TMax}{p} \rceil$.
11. On fixe le paramètre entier *xrendement*, par exemple *xrendement* $\in [2, 3]$. Le rendement de production de chaque période est fixé aléatoirement dans l'intervalle $[1, \frac{C^{Tank}}{xrendement}]$. L'idée ici est d'imposer qu'il faudrait au moins produire sur deux périodes pour pouvoir remplir la citerne.
12. On fixe le paramètre entier *xcout*, par exemple *xcout* $\in [1, M]$. Le coût de production de chaque période est fixé aléatoirement dans l'intervalle $[1, xcout]$.

Génération du paquet d'instances INST_CTE

On génère 20 instances avec le procédé de génération décrit par la suite. Avant de le décrire on présente les paramètres du processus de génération :

- K : Le nombre d'intervalles, ainsi les N périodes seront divisées en K intervalles. Pour des raisons de simplicité, nous allons faire en sorte que ces intervalles aient le même nombre de périodes. Nous ajoutons le paramètre n , qui sera le nombre de périodes d'un intervalle. Pour ce motif, N ne sera pas un paramètre et il sera calculé comme $N = n \times K$.
- Q exprime la contrainte induite par les besoins énergétiques (énergie à recharger) du véhicule et la capacité de stockage du véhicule, il s'agit du nombre de recharges prévu.
- H : Le rapport entre la capacité de la citerne et le réservoir du véhicule.
- R exprime la contrainte induite par les besoins énergétiques du véhicule sur l'activité de production.

- S : Le rapport entre la part du coût due à la consommation d'énergie et la part du coût due à l'activation de la micro-usine.

Nous calculons l'énergie minimale attendue E^* comme l'énergie nécessaire dans une tournée simple construite de la manière suivante : en partant du dépôt, nous sélectionnons itérativement la station non visitée la plus proche jusqu'à ce que toutes les stations soient visitées, puis, nous retournons au dépôt final. Nous calculons également l'énergie moyenne pour se rendre à la micro-usine $E_{mean}^{MP} = \frac{1}{M+1} \sum_{0 \leq i \leq M} \varepsilon_i$.

Pour chaque intervalle $k \in K$, nous générerons aléatoirement 2 nombres Π_k avec $k \in 1, \dots, 5$ et Θ_k avec $k \in 1, \dots, 3$. Pour chaque période i de l'intervalle k , nous générerons aléatoirement un taux de production $RAux_i$ et un coût de production variable $Cost_i^V$ selon la distribution uniforme respectivement à l'intérieur des intervalles $[\Pi_k/2, 3\Pi_k/2]$ et $[\Theta_k/2, 3\Theta_k/2]$. Ensuite, nous obtenons le coût de production $R_i = \lambda \times RAux_i$, en choisissant λ de telle sorte que $\sum_i R_i = R \times E^*$. Nous calculons le coût moyen unitaire : $C_{mean} = (\sum_i Cost_i^V / R_i) \times N$.

Nous fixons $C^{Veh} = E^*/Q + E_{mean}^{MP}$, nous ajoutons E_{mean}^{MP} pour prendre en compte le fait que le véhicule a également besoin de capacité pour aller se ravitailler en carburant et pour les bonnes tournées, le nombre de recharges est approximativement Q . Nous fixons $C^{Tank} = H \times C^{Veh}$ et nous fixons le coût de production fixe : $Cost^F = \frac{C_{mean} \times E^*}{2S \times (1 + E^*/C^{Veh})}$.

4.7.3 Caractéristiques des instances

Toutes les séries d'expériences concernent les instances présentées dans les tableaux (4.3) et (4.4) avec les notations suivantes :

- num : Numéro de l'instance ;
- M : Nombre de stations ;
- N : Nombre de périodes ;
- p : Durée d'une période ;
- H_0 : Quantité d'hydrogène initiale dans la citerne ;
- C^{Tank} : Capacité maximale de la citerne ;
- E_0 : Quantité d'hydrogène initiale dans le réservoir du véhicule ;
- C^{Veh} : Capacité maximale du réservoir ;
- $LTour$: Durée de la tournée sans recharge ;
- $ETour$: Énergie de la tournée sans recharge.

Les tableaux (4.3) et (4.4) représentent respectivement les caractéristiques du paquet d'instances INST_VAR et celles du paquet d'instances INST_CTE. On fait les remarques suivantes :

- Pour la plupart des instances du paquet INST_VAR, l'énergie initial E_0 du véhicule est très petite par rapport à l'énergie de la tournée sans recharge $ETour$. Or cela n'est pas le cas pour les instances du paquet INST_CTE ;
- Pour la plupart des instances du paquet INST_VAR, l'énergie initial E_0 du véhicule est très petite par rapport à la capacité du véhicule C^{Veh} . Or cela n'est pas le cas pour les instances du paquet INST_CTE ; on fait la même remarque pour H_0 par rapport à C^{Tank} ;

- Les capacités du véhicule et de la citerne du paquet d'instances INST_VAR sont très élevées par rapport à celles du paquet d'instances INST_CTE;
- Le nombre de périodes N des instances du paquet INST_CTE est constant, or, dans le paquet d'instances INST_VAR on a différentes valeurs de N ;
- Pour la plupart des instances, les valeurs de N du paquet d'instances INST_VAR sont très élevées par rapport à celles du paquet d'instances INST_CTE;
- Pour la plupart des instances, les valeurs de p du paquet d'instances INST_VAR sont très faibles par rapport à celles du paquet d'instances INST_CTE.

Caractéristiques des instances du paquet INST_VAR

Le tableau (4.3) présente les caractéristiques des instances du paquet INST_VAR. Le paquet d'instances INST_VAR est constitué de 30 instances. On a des instances de 8, 10, 12, 15, 20, 30, 50, 70 et 100 stations. La durée d'une période vaut 1, 2, 4, 5, 6, 8 et 10.

num	M	N	p	H ₀	C ^{Tank}	E ₀	C ^{Veh}	LTour	Etour
1	8	20	4	6	25	8	12	20	20
2	8	25	4	5	20	8	10	25	26
3	8	40	5	10	70	20	30	44	50
4	10	36	2	8	25	9	12	36	38
5	10	50	4	6	40	10	20	50	54
6	10	94	1	3	30	4	15	47	52
7	12	32	4	0	50	4	25	63	68
8	12	50	4	3	36	3	18	50	58
9	15	160	4	8	240	20	120	426	556
10	20	108	10	20	390	10	190	716	894
11	10	80	2	10	40	10	20	38	38
12	15	327	4	20	400	11	140	653	790
13	20	180	6	3	500	20	170	718	834
14	20	440	5	100	350	50	170	910	1164
15	30	177	8	6	420	10	200	944	1188
16	30	260	6	50	400	50	200	1034	1320
17	30	544	4	10	560	15	250	1088	1382
18	50	265	10	6	400	9	200	1764	2216
19	50	500	8	100	420	50	180	1565	2000
20	70	328	10	15	550	25	270	2183	2766
21	30	1100	2	100	400	80	200	1097	1342
22	30	1200	2	100	400	70	200	957	1200
23	50	634	6	50	300	50	150	1542	1938
24	50	850	4	20	400	50	180	1941	2500
25	50	1125	4	20	700	50	280	1718	2128
26	70	383	8	20	760	15	240	2039	2590
27	70	683	8	8	850	14	370	2185	2826
28	70	984	6	50	750	40	250	2340	2926
29	100	651	8	100	600	60	200	3471	4360
30	100	800	10	100	950	100	430	3095	3796

TABLE 4.3 – Caractéristiques des instances du paquet INST_VAR

Caractéristiques des instances du paquet INST_CTE

Le tableau (4.4) présente les caractéristiques des instances du paquet INST_CTE. Le paquet d'instances INST_CTE est constitué de 20 instances. On a des instances de 10, 20, 30, 50, 75 et 100 stations.

Le nombre de périodes est fixe. Le nombre de périodes est 20 pour toutes les instances. La durée d'une période vaut 5, 10, 14, 34, 58 et 65.

num	M	N	p	H ₀	C ^{Tank}	E ₀	C ^{Veh}	LTour	Etour
31	10	20	5	51	73	34	49	48	54
32	10	20	5	48	69	32	46	50	56
33	10	20	5	42	60	28	40	47	54
34	10	20	5	54	78	36	52	46	52
35	20	20	10	65	94	32	47	84	100
36	20	20	10	72	104	36	52	73	78
37	20	20	10	42	60	42	60	78	82
38	20	20	10	35	51	35	51	101	112
39	20	20	10	34	49	34	49	73	80
40	30	20	14	85	122	85	122	136	156
41	50	20	34	65	93	43	62	218	238
42	50	20	34	56	81	37	54	205	224
43	75	20	58	102	147	41	59	275	308
44	75	20	58	109	157	44	63	217	232
45	100	20	65	198	284	49	71	265	278
46	100	20	65	232	332	58	83	283	298
47	100	20	65	233	333	77	111	274	284
48	100	20	65	256	366	85	122	386	424
49	100	20	65	203	291	67	97	253	260
50	100	20	65	165	237	55	79	256	268

TABLE 4.4 – Caractéristiques des instances du paquet INST_CTE

4.7.4 Caractéristiques des solutions

Dans cette section, on présente les caractéristiques des solutions des 50 instances générées. Ces solutions ont été obtenues par le modèle $MILP_{SMEPC}$ en y ajoutant les contraintes (STC), (EC1), (EC2) et (EC3). Lorsque la solution optimale n'est pas connue, on donne les caractéristiques de la borne supérieure. La liste de ces caractéristiques est :

- nbRech : Nombre de recharges ;
- nbSetup : Nombre d'activations de la micro-usine
- prod : Hydrogène produite pour réaliser la tournée ;
- conso : Hydrogène consommée par le véhicule pour réaliser la tournée ;
- waittime : Temps d'attente du véhicule ;
- costtime : Coût en temps de la solution ;
- costprod : Coût de production de la solution.

En moyenne, on constate que pour toutes les instances, la durée de la tournée **costtime** est très élevée par rapport à la durée de la tournée sans recharge **LTour**, ce qui peut vouloir dire que le véhicule attend « beaucoup » avant de se recharger ou que la durée des détours est très élevée. On remarque aussi qu'il n'y a pas un grand gap entre la quantité d'hydrogène produite et la quantité d'hydrogène consommée, ce qui peut vouloir dire qu'on ne produit pas de façon excessive.

Les tableaux (4.5) et (4.6) représentent respectivement les caractéristiques des solutions du paquet d'instances INST_VAR et celles du paquet d'instances INST_CTE. On fait les remarques suivantes :

- Concernant les instances du paquet INST_VAR, le temps d'attente **waittime** du véhicule est très faible par rapport au coût en temps **costtime** du processus, on passe en moyenne 58,62% du temps à attendre. Or cela n'est pas le cas pour les instances du paquet INST_CTE ;

- Globalement, le nombre de recharge du paquet d'instances INST_VAR est supérieur au nombre de recharge du paquet d'instances INST_CTE, cela est peut-être dû au fait que les valeurs E_0 du paquet INST_VAR sont très petites par rapport aux valeurs E_{tour} ;
- Concernant les instances du paquet INST_VAR, le coût de production $costprod$ est très faible par rapport au coût en temps $costtime$. Or cela n'est pas le cas pour les instances du paquet INST_CTE.
- La quantité d'hydrogène produite et consommée du paquet d'instances INST_VAR est très élevée par rapport à celle du paquet d'instances INST_CTE ;

Caractéristiques des solutions des instances du paquet INST_VAR

Le tableau (4.5) présente les caractéristiques des solutions des instances du paquet INST_VAR.

num	nbRech	nbSetup	prod	conso	waittime	costtime	costprod
1	3	2	29	28	30	57	74
2	5	2	34	34	40	73	78
3	3	2	64	62	25	81	63
4	5	2	42	42	26	67	73
5	5	2	63	60	29	85	76
6	7	4	72	72	8	73	105
7	5	3	92	90	39	121	101
8	6	2	69	66	58	121	71
9	7	4	624	622	40	509	135
10	8	4	1104	1084	133	1001	138
11	4	2	61	58	17	75	59
12	10	3	952	946	67	819	93
13	7	3	975	972	70	867	89
14	12	5	1442	1412	89	1231	141
15	8	5	1354	1328	107	1196	150
16	9	4	1391	1390	76	1160	131
17	8	3	1474	1424	50	1182	103
18	15	8	2400	2400	215	2125	248
19	17	6	2211	2210	187	1977	236
20	13	7	2851	2848	207	2449	210
21	11	5	1545	1488	28	1279	131
22	11	4	1500	1456	27	1221	121
23	22	11	2510	2510	178	2276	314
24	22	11	3036	3022	111	2517	288
25	12	5	2293	2278	65	1924	129
26	15	5	2617	2610	155	2244	112
27	11	5	2833	2828	113	2317	151
28	18	6	3374	3360	164	2879	175
29	30	10	4626	4616	411	4129	308
30	12	6	3871	3864	196	3362	160

TABLE 4.5 – Caractéristiques des solutions des instances du paquet INST_VAR

Caractéristiques des solutions des instances du paquet INST_CTE

Le tableau (4.6) présente les caractéristiques des solutions des instances du paquet INST_CTE.

num	nbRech	nbSetup	prod	conso	waittime	costtime	costprod
31	2	2	59	56	16	66	175
32	3	2	64	64	23	81	277
33	2	2	56	56	11	61	162
34	2	2	54	54	11	61	141
35	4	2	121	114	53	151	106
36	3	1	98	92	57	141	85
37	2	2	87	84	32	111	66
38	4	4	122	120	62	171	111
39	4	4	118	114	47	151	162
40	2	4	169	160	38	177	159
41	6	6	290	290	355	613	288
42	7	6	283	280	363	613	827
43	9	8	464	402	743	1103	433
44	6	3	294	294	485	755	164
45	6	6	385	380	453	794	135
46	5	3	338	338	466	781	70
47	4	3	409	358	379	716	226
48	5	3	517	502	598	1041	209
49	4	3	340	340	394	716	115
50	6	6	385	372	503	846	234

TABLE 4.6 – Caractéristiques des solutions des instances du paquet INST_CTE

4.7.5 Résultats du modèle $MILP_{SMEPC}$ avec temps limite de 3 heures et 8 threads

Cette section présente les résultats du modèle $MILP_{SMEPC}$ avec les contraintes (STC), (EC1), (EC2) et (EC3). Pour chaque instance, on donne la borne inférieure qu'on nomme $BInf$, la borne supérieure qu'on nomme $BSup$ et le temps CPU en secondes obtenue par CPLEX. Ces valeurs $BSup$ seront utilisées comme valeurs de références pour calculer les gaps dans tout ce document. $gapMI$ est le gap entre la borne inférieure et la borne supérieure. Des essais préliminaires ont été effectués avec la formulation de base de la section 4.4 uniquement. Sa faiblesse est rapidement apparue puisque CPLEX n'a pas pu trouver une solution faisable à la majorité des instances après une heure de temps CPU. Nous observons alors qu'avec les contraintes *Simple Time Constraints* (4.8a) et (4.8b), CPLEX peut atteindre une solution faisable sur plus de la moitié de ces instances. Ceci explique pourquoi la formulation de référence $MILP_{SMEPC}$ intégrera les contraintes (STC). Dans le cadre de la résolution en nombres entiers du modèle de référence, les tableaux (4.7) et (4.8) montrent que 38 instances sur les 50 testées sont résolues de manière optimale. On fait les remarques suivantes :

- On trouve la solution optimale de toutes les instances du paquet INST_CTE, or on trouve la solution optimale de 18 instances du paquet INST_VAR mais pour les 12 instances restantes on obtient moins de 5% de gap entre $BSup$ et $BInf$;
- En moyenne, L'exécution des instances INST_CTE est beaucoup plus rapide que celle de INST_VAR.

Résultats du modèle $MILP_{SMEPC}$ pour les instances du paquet INST_VAR

Le tableau (4.7) présente les résultats du modèle $MILP_{SMEPC}$ avec les contraintes (STC), (EC1), (EC2) et (EC3) pour les instances du paquet INST_VAR. Concernant les 30 instances du paquet INST_VAR, le modèle $MILP_{SMEPC}$ fournit la solution optimale de 18 instances en moins de 3 heures sur 8 threads.

num	BInf	BSup	CPU	gapMI
1	131	131	0,31	0,00
2	151	151	2,22	0,00
3	144	144	1,91	0,00
4	140	140	2,51	0,00
5	161	161	1,87	0,00
6	178	178	7,09	0,00
7	222	222	0,77	0,00
8	192	192	9,94	0,00
9	644	644	3,05	0,00
10	1139	1139	4,55	0,00
11	134	134	2,16	0,00
12	912	912	43,36	0,00
13	956	956	3,04	0,00
14	1372	1372	39,92	0,00
15	1346	1346	48,54	0,00
16	1291	1291	179,01	0,00
17	1285	1285	763,64	0,00
18	2372,45	2373	6708,77	0,02
19	2190,02	2213	10802,20	1,04
20	2562,58	2659	10802,40	3,63
21	1393,95	1410	10806,10	1,14
22	1334,71	1342	10804,70	0,54
23	2556,17	2590	10803,40	1,31
24	2763,18	2805	10804,00	1,49
25	2053	2053	1165,38	0,00
26	2343,35	2356	10805,60	0,54
27	2449,41	2468	10803,10	0,75
28	2944,06	3054	10803,90	3,60
29	4246,45	4437	10803,10	4,29
30	3410,42	3522	10803,60	3,17

TABLE 4.7 – Résultats du modèle $MILP_{SMEPC}$ sur les instances du paquet INST_VAR. Le temps limite est 3 heures sur **8 threads**.

Résultats du modèle $MILP_{SMEPC}$ pour les instances du paquet INST_CTE

Le tableau (4.8) présente les résultats du modèle $MILP_{SMEPC}$ avec les contraintes (STC), (EC1), (EC2) et (EC3) pour les instances du paquet INST_CTE. On obtient les solutions optimales des 20 instances du paquet INST_CTE avec le modèle $MILP_{SMEPC}$ en moins de 3 heures sur **8 threads**.

num	BInf	BSup	CPU
31	241	241	0,414
32	358	358	0,443
33	223	223	0,163
34	202	202	0,266
35	257	257	0,66
36	226	226	0,614
37	177	177	0,643
38	282	282	0,602
39	313	313	1,62
40	336	336	1,197
41	901	901	9,169
42	1440	1440	3,091
43	1536	1536	29,316
44	919	919	13,367
45	929	929	50,343
46	851	851	21,56
47	942	942	10,719
48	1250	1250	27,978
49	831	831	53,674
50	1080	1080	39,143

TABLE 4.8 – Résultats du modèle $MILP_{SMEPC}$ sur les instances du paquet INST_CTE. Le temps limite est 3 heures sur 8 threads.

4.7.6 Résultats des modèles $RMILP_{SMEPC}$ et $MILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread

Nous présentons ici les résultats de tous les modèles sur les instances des paquets INST_CTE et INST_VAR. Pour les résultats suivants, CPLEX 12.10 est utilisé en mode *single-threads*. Le temps maximum du CPU est fixé à 1 heure.

Résultats du $RMILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread

Dans cette section, les expérimentations concernent la relaxation linéaire $RMILP_{SMEPC}$. Nous étudions la contribution des inégalités données dans la section 4.5. Dans chaque cas, on donne la valeur optimale de la fonction objectif, le temps CPU et le gap à la solution optimale si elle existe ou à la meilleure borne, sinon. Les tableaux (4.9), (4.10), (4.11) et (4.12) présentent les résultats pour :

- (LR) : $RMILP_{SMEPC}$ la relaxation linéaire seule ;
- (LR)+(STC) : $RMILP_{SMEPC}$ lorsque seules les contraintes (STC) sont ajoutées ;
- (EC1+EC2) : $RMILP_{SMEPC}$ lorsque les contraintes (EC1) et (EC2) sont ajoutées à (LR)+(STC) ;
- (EC3) : $RMILP_{SMEPC}$ lorsque les contraintes (EC3) sont ajoutées à (LR)+(STC) ;
- All energy : $RMILP_{SMEPC}$ lorsque les contraintes (EC1), (EC2) et (EC3) sont ajoutées à (LR)+(STC).

Pour chaque instance, les résultats suivants sont spécifiés :

- CPU : Temps total du CPU en secondes ;
- Obj : Valeur optimale du programme linéaire ;
- $GapF = 100 \times \frac{opt-Obj}{opt}$: Pourcentage de l'erreur relative entre la meilleure solution réalisable opt (solution optimale si le problème a été résolu jusqu'à l'optimalité) et la valeur optimale de

la fonction objectif de la relaxation linéaire Obj . Si une instance n'a pas été résolue jusqu'à l'optimalité, on utilise la borne supérieure de l'exécution sur 8 *threads* avec un temps limite de 3 heures (voir tableau (4.7) et (4.8)). Les instances indiquées par "-" sont celles dont le temps de calcul de la solution optimale a dépassé 1h.

Les tableaux (4.9) et (4.10) présentent les résultats de la relaxation linéaire $RMILP_{SMEPC}$ du paquet d'instances INST_VAR. Pareillement, les tableaux (4.11) et (4.12) présentent les résultats de la relaxation linéaire $RMILP_{SMEPC}$ du paquet d'instances INST_CTE. On constate que les contraintes (STC) sont les plus efficaces car ce sont elles qui améliore le plus le gap surtout sur les grandes instances. Mais elles induisent une légère augmentation du temps CPU. De plus, Les contraintes (EC) ont en général sur toutes les instances un faible impact sur la gap. On fait les remarques suivantes :

- L'ajout de la contrainte (STC) pour les instances du paquet INST_VAR induit un plus grand écart que pour le paquet INST_CTE parce que comme on l'a dit précédemment les solutions des instances INST_VAR ont un coût temps très grand par rapport au coût production. C'est pour cette raison que l'amélioration de la partie temps a une grande incidence sur le coût final;
- Les solutions obtenues pas ces modèles sur le paquet d'instances INST_VAR sont meilleures que celles obtenues sur les instances INST_CTE. Donc ces modèles marchent mieux sur les instances INST_VAR ;
- Le temps de calcul du paquet d'instances INST_VAR est supérieur au temps de calcul des instances INST_CTE.

num	LR			LR + STC		
	Obj	CPU	GapF	Obj	CPU	GapF
1	44,15	0,0	66,29	68,18	0,0	47,95
2	36,94	0,0	75,53	64,89	0,1	57,02
3	30,93	0,1	78,52	79,51	0,1	44,79
4	45,34	0,0	67,62	82,52	0,0	41,06
5	46,42	0,1	71,17	97,27	0,0	39,58
6	41,26	0,1	76,82	97,79	0,1	45,06
7	52,00	0,1	76,57	124,84	0,0	43,77
8	39,97	0,0	79,18	96,43	0,0	49,78
9	80,16	0,1	87,55	523,61	0,1	18,69
10	59,41	0,1	94,78	883,52	0,1	22,43
11	27,60	0,1	79,40	78,80	0,0	41,19
12	37,90	0,2	95,84	717,05	0,4	21,38
13	33,34	0,1	96,51	794,71	0,2	16,87
14	69,60	0,4	94,93	1049,27	1,3	23,52
15	68,84	0,2	94,89	1114,54	0,3	17,20
16	62,16	0,3	95,19	1120,57	0,4	13,20
17	49,10	0,5	96,18	1155,17	2,4	10,10
18	125,35	0,6	94,72	1942,44	0,8	18,14
19	137,60	1,4	93,78	1826,41	5,0	17,47
20	106,04	1,0	96,01	2311,54	1,5	13,07
21	70,42	3,3	95,01	1240,39	10,6	12,03
22	60,27	2,7	95,51	1106,55	7,0	17,54
23	134,63	2,2	94,80	1910,07	7,6	26,25
24	135,81	3,2	95,16	2268,60	10,8	19,12
25	75,16	7,3	96,34	1864,20	17,3	9,20
26	15,78	2,8	99,33	2088,61	4,0	11,35
27	76,73	2,9	96,89	2268,94	7,7	8,07
28	90,08	7,3	97,05	2661,38	21,2	12,86
29	175,85	6,2	96,04	3759,44	13,9	15,27
30	72,49	3,4	97,94	3191,14	8,4	9,39

TABLE 4.9 – Paquet d'instances INST_VAR : la relaxation linéaire $RMILP_{SMEPC}$ et les effets des contraintes STC

num	LR + STC			EC1+EC2			EC3			All Energy		
	Obj	CPU	GapF	Obj	CPU	GapF	Obj	CPU	GapF	Obj	CPU	GapF
1	68,18	0,0	47,95	68,18	0,1	47,95	79,5258	0,1	39,29	81,77	0,0	37,58
2	64,89	0,1	57,02	77,25	0,0	48,84	73,6146	0,1	51,25	87,58	0,1	42,00
3	79,51	0,1	44,79	85,18	0,1	40,85	80,7294	0,1	43,94	86,48	0,1	39,94
4	82,52	0,0	41,06	95,13	0,0	32,05	89,1597	0,1	36,31	103,15	0,1	26,32
5	97,27	0,0	39,58	102,97	0,1	36,04	102,194	0,1	36,53	108,52	0,1	32,60
6	97,79	0,1	45,06	117,31	0,1	34,09	115,567	0,1	35,07	134,76	0,1	24,29
7	124,84	0,0	43,77	136,46	0,0	38,53	142,594	0,1	35,77	155,67	0,1	29,88
8	96,43	0,0	49,78	109,33	0,1	43,06	110,698	0,1	42,34	125,99	0,1	34,38
9	523,61	0,1	18,69	536,21	0,1	16,74	570,346	0,2	11,44	584,29	0,3	9,27
10	883,52	0,1	22,43	909,99	0,1	20,11	963,463	0,2	15,41	984,70	0,3	13,55
11	78,80	0,0	41,19	88,46	0,0	33,99	91,3411	0,1	31,84	102,31	0,1	23,65
12	717,05	0,4	21,38	732,63	0,2	19,67	830,481	0,9	8,94	846,13	0,7	7,22
13	794,71	0,2	16,87	807,91	0,2	15,49	854,829	0,4	10,58	863,65	0,2	9,66
14	1049,27	1,3	23,52	1073,81	0,5	21,73	1218,37	2,8	11,20	1246,31	1,8	9,16
15	1114,54	0,3	17,20	1135,81	0,2	15,62	1183,52	0,4	12,07	1200,46	0,5	10,81
16	1120,57	0,4	13,20	1149,03	0,4	11,00	1160,43	1,7	10,11	1189,91	1,7	7,83
17	1155,17	2,4	10,10	1172,22	1,2	8,78	1196,6	3,9	6,88	1209,30	7,0	5,89
18	1942,44	0,8	18,14	1995,63	0,8	15,90	2059,16	1,1	13,23	2110,38	1,7	11,07
19	1826,41	5,0	17,47	1856,81	3,7	16,10	1902,51	11,0	14,03	1932,25	21,6	12,69
20	2311,54	1,5	13,07	2351,12	0,9	11,58	2342,78	7,1	11,89	2382,36	16,3	10,40
21	1240,39	10,6	12,03	1263,48	2,3	10,39	1316,83	18,7	6,61	1340,30	7,6	4,94
22	1106,55	7,0	17,54	1129,40	4,4	15,84	1228,15	20,0	8,48	1252,20	34,8	6,69
23	1910,07	7,6	26,25	1971,99	4,5	23,86	2316,46	8,7	10,56	2387,33	12,3	7,83
24	2268,60	10,8	19,12	2325,79	2,4	17,08	2561,57	16,7	8,68	2621,32	14,1	6,55
25	1864,20	17,3	9,20	1876,85	3,9	8,58	1944,81	42,0	5,27	1956,51	83,2	4,70
26	2088,61	4,0	11,35	2151,98	1,0	8,66	2115,79	8,3	10,20	2176,70	8,3	7,61
27	2268,94	7,7	8,07	2289,61	6,2	7,23	2292,71	17,3	7,10	2309,62	27,9	6,42
28	2661,38	21,2	12,86	2685,29	15,3	12,07	2815,69	52,5	7,80	2839,81	119,2	7,01
29	3759,44	13,9	15,27	3813,40	12,1	14,05	3914,15	30,8	11,78	3970,15	101,5	10,52
30	3191,14	8,4	9,39	3224,40	7,6	8,45	3233,22	8,0	8,20	3265,79	64,1	7,27

TABLE 4.10 – Paquet d’instances INST_VAR : la relaxation linéaire $RMILP_{SMEPC}$ et les effets des contraintes EC

num	LR			LR + STC		
	Obj	CPU	GapF	Obj	CPU	GapF
31	108,66	0,20	54,91	157,79	0,16	34,53
32	217,93	0,16	39,13	269,88	0,23	24,62
33	138,71	0,20	37,80	186,87	0,16	16,20
34	104,03	0,10	48,50	151,90	0,06	24,80
35	79,16	0,20	69,20	156,41	0,17	39,14
36	50,91	0,20	77,47	128,14	0,16	43,30
37	50,04	0,19	71,73	128,71	0,20	27,28
38	78,96	0,30	72,00	183,29	0,06	35,00
39	71,15	0,20	77,27	147,99	0,21	52,72
40	116,70	0,20	65,27	252,87	0,06	24,74
41	210,73	0,26	76,61	448,75	0,40	50,19
42	272,13	0,26	81,10	479,29	0,33	66,72
43	178,47	0,51	88,38	482,01	0,67	68,62
44	61,71	0,19	93,28	298,81	0,47	67,49
45	49,54	0,19	94,67	348,72	0,17	62,46
46	39,49	0,18	95,36	330,94	0,21	61,11
47	61,65	0,19	93,46	359,46	0,22	61,84
48	108,60	0,43	91,31	508,19	0,31	59,35
49	56,17	0,36	93,24	332,13	0,23	60,03
50	55,96	0,17	94,82	338,53	0,23	68,65

TABLE 4.11 – Paquet d’instances INST_CTE : la relaxation linéaire $RMILP_{SMEPC}$ et les effets des contraintes STC

num	LR + STC			EC1+EC2			EC3			All Energy		
	Obj	CPU	GapF	Obj	CPU	GapF	Obj	CPU	GapF	Obj	CPU	GapF
31	157,79	0,16	34,53	157,79	0,27	34,53	160,85	0,16	33,26	160,85	0,16	33,26
32	269,88	0,23	24,62	269,93	0,23	24,60	278,24	0,08	22,28	278,24	0,16	22,28
33	186,87	0,16	16,20	186,87	0,22	16,20	191,85	0,16	13,97	191,85	0,17	13,97
34	151,90	0,06	24,80	152,50	0,22	24,50	152,19	0,08	24,66	152,80	0,16	24,36
35	156,41	0,17	39,14	168,41	0,25	34,47	166,01	0,08	35,41	178,92	0,37	30,38
36	128,14	0,16	43,30	141,22	0,22	37,51	129,16	0,15	42,85	142,35	0,15	37,01
37	128,71	0,20	27,28	128,85	0,22	27,20	129,94	0,18	26,59	130,08	0,20	26,51
38	183,29	0,06	35,00	183,74	0,23	34,84	190,27	0,21	32,53	190,83	0,26	32,33
39	147,99	0,21	52,72	150,56	0,22	51,90	183,77	0,20	41,29	185,44	0,26	40,76
40	252,87	0,06	24,74	252,87	0,23	24,74	252,92	0,19	24,73	252,92	0,20	24,73
41	448,75	0,40	50,19	448,75	0,70	50,19	473,06	0,56	47,50	473,06	0,73	47,50
42	479,29	0,33	66,72	702,06	0,42	51,25	558,83	0,70	61,19	810,72	0,83	43,70
43	482,01	0,67	68,62	508,52	1,47	66,89	517,78	1,18	66,29	544,93	1,98	64,52
44	298,81	0,47	67,49	311,84	0,44	66,07	349,50	0,37	61,97	363,01	0,90	60,50
45	348,72	0,17	62,46	349,00	0,85	62,43	396,89	0,45	57,28	397,57	1,02	57,20
46	330,94	0,21	61,11	330,94	0,77	61,11	335,75	0,56	60,55	335,75	0,96	60,55
47	359,46	0,22	61,84	373,94	0,65	60,30	389,35	0,32	58,67	405,59	0,97	56,94
48	508,19	0,31	59,35	509,54	1,31	59,24	533,02	0,76	57,36	534,59	1,92	57,23
49	332,13	0,23	60,03	332,13	1,08	60,03	378,90	0,56	54,40	378,90	1,37	54,40
50	338,53	0,23	68,65	345,95	0,81	67,97	441,05	0,56	59,16	442,53	1,29	59,02

TABLE 4.12 – Paquet d’instances INST_CTE : la relaxation linéaire $RMILP_{SMEPC}$ et les effets des contraintes EC

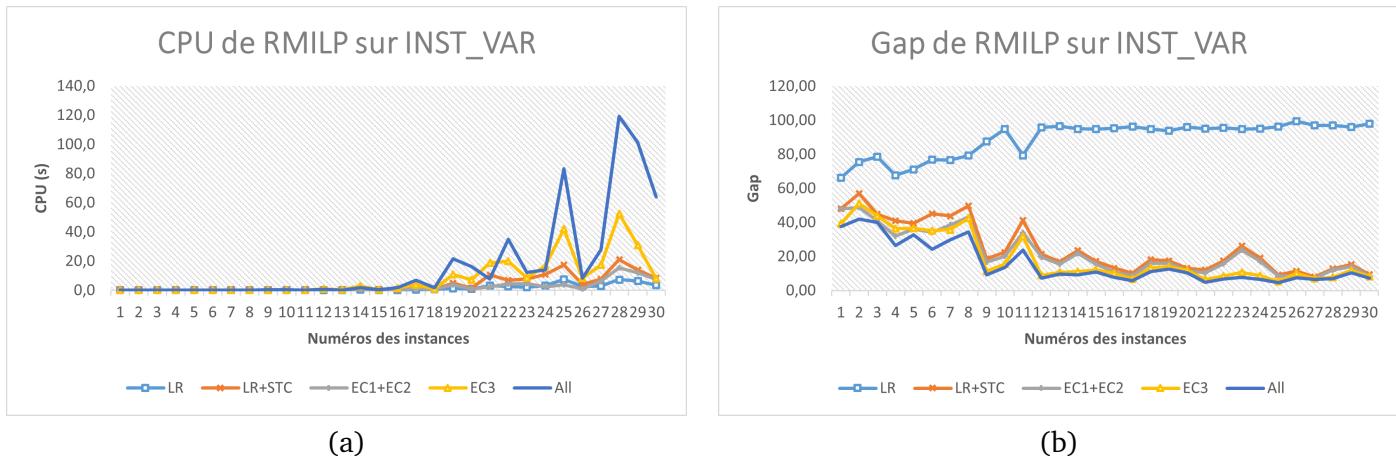


FIGURE 4.10 – Représentation graphique des tableaux (4.9) et (4.10). (a) représente le temps CPU et (b) représente le gap à l'optimalité de chaque instance de INST_VAR.

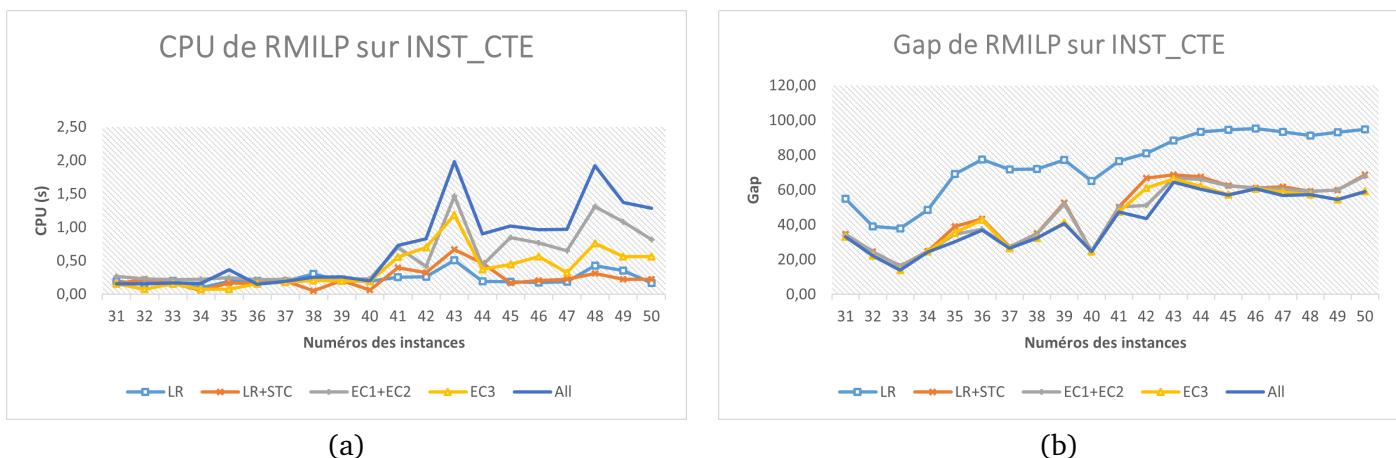


FIGURE 4.11 – Représentation graphique des tableaux (4.11) et (4.12). (a) représente le temps CPU et (b) représente le gap à l'optimalité de chaque instance de INST_CTE.

Résultats du $MILP_{SMEPC}$ avec comme temps limite 1 heure sur 1 thread

Les tableaux (4.13), (4.14), (4.15) et (4.16) présentent les résultats du modèle $MILP_{SMEPC}$. La signification des colonnes est :

- (STC) : $MILP_{SMEPC}$ lorsque seules les contraintes (STC) sont ajoutées ;
- (EC1+EC2) : $MILP_{SMEPC}$ lorsque les contraintes (EC1) et (EC2) sont ajoutées à (STC) ;
- (EC3) : $MILP_{SMEPC}$ lorsque les contraintes (EC3) sont ajoutées à (STC) ;
- All energy : $MILP_{SMEPC}$ lorsque les contraintes (EC1), (EC2) et (EC3) sont ajoutées à (STC) .

Pour chaque instances, les résultats suivants sont spécifiés :

- CPU : temps total du CPU en secondes ;
- BInf et BSup : Respectivement la borne inférieure et la borne supérieure du programme linéaire mixte en nombres entiers. La borne supérieure correspond à la meilleure solution entière réalisable ;
- $gapMI = 100 \times \frac{BSup - BInf}{BSup}$: pourcentage de l'erreur relative entre la borne inférieure et la borne supérieure ; Les instances indiquées par "-" sont celles dont le temps CPU a dépassé 1h et aucune borne supérieure n'a été obtenue.

Les tableaux (4.13) et (4.14) présentent les résultats du modèle $MILP_{SMEPC}$ sur le paquet d'instances INST_VAR. Pareillement, les tableaux (4.15) et (4.16) présentent les résultats du modèle $MILP_{SMEPC}$ sur le paquet d'instances INST_CTE. Pour le paquet d'instances INST_CTE, on remarque que lorsque les contraintes (EC3) sont ajoutées à (STC) cela entraîne une importante diminution du temps CPU. De plus, on a une amélioration du gap. Or, (EC3) a un plus faible impact sur les instances du paquet INST_VAR.

STC				
num	BInf	BSup	CPU	gapMI
1	131	131	1,0	0,00
2	151	151	2,3	0,00
3	144	144	2,3	0,00
4	140	140	3,9	0,00
5	161	161	1,7	0,00
6	178	178	33,8	0,00
7	222	222	3,5	0,00
8	192	192	25,5	0,00
9	644	644	17,7	0,00
10	1139	1139	37,7	0,00
11	134	134	7,6	0,00
12	912	912	458,2	0,00
13	956	956	48,6	0,00
14	1372	1372	2629,5	0,00
15	1346	1346	1188,2	0,00
16	1291	1291	1620,9	0,00
17	1246,49	1285	3600,1	3,00
18	2208,43	2456	3600,1	10,08
19	2038,48	2275	3600,1	10,40
20	2416,31	2666	3600,1	9,37
21	1353,44	1424	3600,2	4,96
22	1253,15	1363	3600,3	8,06
23	2237,48	-	3600,1	-
24	2572,50	-	3600,2	-
25	1946,27	-	3600,2	-
26	2175,70	2458	3600,1	11,48
27	2335,89	2507	3600,2	6,83
28	2768,95	4544	3600,5	39,06
29	3941,23	-	3600,3	-
30	3269,36	-	3600,3	-

TABLE 4.13 – Paquet d'instances INST_VAR : $MILP_{SMEPC}$ avec les contraintes STC. Le temps limite est 1 heure en mode *single-threads*.

num	EC1, EC2				EC3				All energy			
	BInf	BSup	CPU	gapMI	BInf	BSup	CPU	gapMI	BInf	BSup	CPU	gapMI
1	131	131	0,5	0	131	131	0,5	0	131	131	0,6	0
2	151	151	1,3	0	151	151	1,9	0	151	151	1,3	0
3	144	144	1,4	0	144	144	1,6	0	144	144	1,3	0
4	140	140	5,8	0	140	140	2,2	0	140	140	2,3	0
5	161	161	2,4	0	161	161	2,3	0	161	161	1,4	0
6	178	178	29,8	0	178	178	64,2	0	178	178	39,9	0
7	222	222	1,7	0	222	222	0,8	0	222	222	0,9	0
8	192	192	33,5	0	192	192	29,4	0	192	192	21,6	0
9	644	644	26,2	0	644	644	63,1	0	644	644	37,4	0
10	1139	1139	26,9	0	1139	1139	35,7	0	1139	1139	8,7	0
11	134	134	7,1	0	134	134	6,7	0	134	134	4,9	0
12	912	912	326,7	0	912	912	259,2	0	912	912	231,4	0
13	956	956	58,1	0	956	956	64,1	0	956	956	32,2	0
14	1372	1372	192,1	0	1372	1372	1686,1	0	1372	1372	885,8	0
15	1346	1346	1321,3	0	1346	1346	1675,0	0	1346	1346	677,5	0
16	1291	1291	2543,8	0	1291	1291	3451,9	0	1291	1291	869,8	0
17	1250,3	1288	3600,1	2,9	1267,4	1285	3600,1	1,4	1272,1	1288	3600,1	1,2
18	2252,2	2408	3600,1	6,5	2317,3	2382	3600,1	2,7	2322,5	2373	3600,1	2,1
19	2044,2	2374	3600,2	13,9	2083,5	2232	3600,1	6,7	2105,9	2247	3600,3	6,3
20	2446,7	2993	3600,2	18,3	2433,7	2713	3600,1	10,3	2462,3	2780	3600,2	11,4
21	1364,4	1412	3600,3	3,4	1363,6	1412	3600,2	3,4	1377,0	-	3600,2	-
22	1288,6	1352	3600,2	4,7	1297,9	1342	3600,2	3,3	1310,7	-	3600,3	-
23	2417,8	2648	3600,2	8,7	2486,8	2591	3600,2	4,0	2502,8	-	3600,2	-
24	2635,4	3132	3600,3	15,9	2682,3	2820	3600,2	4,9	2723,4	2861	3600,3	4,8
25	1975,9	2131	3600,4	7,3	2010,3	2061	3600,3	2,5	2020,0	2057	3600,4	1,8
26	2245,6	2396	3600,2	6,3	2250,9	2372	3600,1	5,1	2299,8	2380	3600,2	3,4
27	2349,8	-	3600,4	-	2350,0	2531	3600,3	7,2	2349,6	-	3600,4	-
28	2812,6	-	3600,5	-	2846,3	-	3600,3	-	2870,3	-	3600,6	-
29	3953,3	-	3600,5	-	4044,9	-	3600,3	-	4113,5	-	3600,5	-
30	3291,8	-	3600,8	-	3287,4	5237	3600,4	37,2	3312,0	-	3600,8	-

TABLE 4.14 – Paquet d'instances INST_VAR : $MILP_{SMEPC}$ avec les contraintes EC. Le temps limite est 1 heure en mode *single-threads*

STC				
num	BInf	BSup	CPU	gapMI
31	241	241	5,38	0
32	358	358	3,78	0
33	223	223	4,04	0
34	202	202	3,98	0
35	257	257	6,96	0
36	226	226	7,00	0
37	177	177	6,30	0
38	282	282	6,99	0
39	313	313	15,00	0
40	336	336	11,31	0
41	901	901	53,51	0
42	1440	1440	45,04	0
43	1536	1536	1105,18	0
44	601,238	1061	3602,28	43,33
45	756,208	929	3601,35	18,60
46	606,157	875	3601,40	30,72
47	942	942	2160,71	0
48	1250	1250	2616,81	0
49	831	831	1097,95	0
50	713,863	1239	3601,71	42,38

TABLE 4.15 – Paquet d'instances INST_CTE : $MILP_{SMEPC}$ avec les contraintes (STC). Le temps limite est 1 heure en mode *single-threads*

num	EC1, EC2				EC3				All energy			
	BInf	BSup	CPU	gapMI	BInf	BSup	CPU	gapMI	BInf	BSup	CPU	gapMI
31	241	241	0,7	0	241	241	0,9	0	241	241	1,2	0
32	358	358	0,4	0	358	358	0,5	0	358	358	0,6	0
33	223	223	0,4	0	223	223	0,5	0	223	223	0,9	0
34	202	202	0,6	0	202	202	0,7	0	202	202	1,0	0
35	257	257	5,8	0	257	257	2,6	0	257	257	3,3	0
36	226	226	2,9	0	226	226	1,8	0	226	226	2,3	0
37	177	177	2,1	0	177	177	1,5	0	177	177	1,5	0
38	282	282	3,9	0	282	282	1,9	0	282	282	3,9	0
39	313	313	19,6	0	313	313	3,1	0	313	313	7,5	0
40	336	336	7,3	0	336	336	7,8	0	336	336	15,0	0
41	901	901	444,7	0	901	901	21,6	0	901	901	108,0	0
42	1440	1440	164,1	0	1440	1440	46,5	0	1440	1440	73,4	0
43	1429,8	1536	3600,1	6,92	1536	1536	75,9	0	1536	1536	1742,7	0
44	592,7	1307	3600,0	54,65	919	919	269,1	0	818,7	938	3600,1	12,7
45	591,2	1078	3600,1	45,16	929	929	332,7	0	768,2	929	3600,2	17,3
46	612,6	882	3600,1	30,55	851	851	353,5	0	670,4	854	3600,1	21,5
47	744,0	942	3600,1	21,02	942	942	291,5	0	942,0	942	2837,6	0
48	864,1	1326	3600,1	34,84	1250	1250	615,0	0	982,0	1250	3600,1	21,4
49	495,7	1411	3600,1	64,87	831	831	158,2	0	831,0	831	1251,3	0
50	540,7	1378	3600,1	60,76	972,3	1096	3600,1	11,3	687,6	1102	3600,1	37,6

TABLE 4.16 – Paquet d'instances INST_CTE : $MILP_{SMEPC}$ avec les contraintes EC. Le temps limite est 1 heure en mode *single-threads*



FIGURE 4.12 – Représentation graphique des tableaux (4.13) et (4.14). (a) représente le temps CPU et (b) représente la valeur gapMI de chaque instance de INST_VAR.



FIGURE 4.13 – Représentation graphique des tableaux (4.15) et (4.16). (a) représente le temps CPU et (b) représente la valeur gapMI de chaque instance de INST_CTE.

Observations et discussions

Une première expérimentation a été faite avec la formulation de base de la section 4.5 uniquement. Sa faiblesse est rapidement apparue puisque CPLEX n'a pas pu trouver une solution faisable à la majorité des instances après une heure de temps CPU. Nous observons alors qu'avec les contraintes (STC), CPLEX peut atteindre une solution faisable sur plus de la moitié des instances. Ceci explique pourquoi la formulation de référence intégrera les contraintes (STC).

On commence par examiner la relaxation linéaire. Des tests sont effectués avec différents ensembles de contraintes pour explorer leur efficacité dans la résolution de ces instances. Dans les tableaux (4.9) et (4.11), la première colonne confirme la déficience de la formulation initiale en raison de l'écart important par rapport à la solution entière optimale. La deuxième colonne nous montre que la valeur de $RMILP_{SMEPC}$ est inférieure à 50% de l'optimum pour la plupart des instances par rapport à la formulation de référence. Le gap avec la meilleure borne supérieure pour les instances difficiles 17 à 30 tombe même à 27 %. Les tableaux (4.10) et (4.12) montrent que ces résultats sont améliorés par l'ajout des contraintes énergétiques. En particulier, les trois ensembles (EC1), (EC2) et (EC3), mis ensemble, donnent les meilleures bornes en contrepartie d'une augmentation modérée du temps d'exécution.

Dans le cadre de la résolution en nombres entiers du modèle de référence, les tableaux (4.13) et (4.14) montrent que seize instances sur les trente testées sont résolues de manière optimale. Ces ensembles de données peuvent être considérés comme des instances *faciles*. Les différents ensembles d'inégalités sont successivement mis en jeu pour tester leurs contributions. On peut noter que dans le tableau (4.14) les contraintes (EC1) et (EC2) n'échouent que sur quatre instances alors que les contraintes (EC3) seules échouent sur deux instances. Les tableaux (4.15) et (4.16) montrent que dix-neuf instances sur les vingt testées sont résolues de manière optimale.

4.8 Conclusion

Dans ce chapitre, on a présenté le problème qu'on cherche à résoudre nommé ici **SMEPC**, ainsi que sa formulation mathématique. Un programme linéaire en nombres entiers mixtes pour le problème SMEPC et plusieurs familles d'inégalités valides sont proposées. Nous avons aussi proposé la relaxation linéaire de ce programme. Les tests expérimentaux montrent la nécessité d'ajouter certaines de ces contraintes afin d'améliorer le modèle de base. Dans le chapitre suivant, on va introduire un algorithme de programmation dynamique global pour résoudre SMEPC.

4.9 Annexes

4.9.1 Démonstration du théorème 2

Cette section présente la démonstration du théorème 2. Considérons une solution optimale $(\bar{z}, \bar{T}, (\bar{L}, \bar{x}))$ de SMEPC, qui décrit respectivement le programme d'activité de la micro-usine, les dates d'arrivée du véhicule aux stations et la politique de recharges en carburant du véhicule. Nous vérifions qu'elle peut être transformée en une solution réalisable du $MILP_{SMEPC}$ avec le même coût.

Soit \bar{J} l'ensemble des indices $j \in \{0, \dots, M\}$ pour lesquels x_j prend la valeur 1. En appliquant les contraintes (4.4a) et (4.4e), la quantité d'hydrogène dans le réservoir du véhicule V^{Veh} est calculée de façon itérative de la manière suivante : si $j \in \bar{J}$, alors $\bar{V}_{j+1}^{Veh} = \bar{V}_j^{Veh} - \varepsilon_j + \bar{L}_j - \varepsilon_{j+1}^*$, sinon $\bar{V}_{j+1}^{Veh} = \bar{V}_j^{Veh} - e_j$.

Comme le véhicule se rend directement à la station suivante après avoir été rechargeé durant une période, \bar{T}^* peut être choisi de telle sorte que :

$$\bar{T}_j^* = \bar{T}_{j+1} - d_{j+1}^* - p \text{ et } \bar{T}_j^* \in \{0, p, \dots, (N-1) \times p\}, \text{ pour } j \in \bar{J}.$$

Lorsque j n'appartient pas à \bar{J} , T_j^* peut prendre n'importe quelle valeur non négative. La validité de la tournée du véhicule est conditionnée par le fait que les vecteurs T et T^* vérifient le bloc de contraintes linéarisées (4.5).

De plus, un indice $\hat{i}(j)$ peut être attribué à chaque j dans \bar{J} , en prenant $\hat{i}(j) \times p = \bar{T}_j^*$. Soit $\bar{I} = \{\hat{i}(j) : j \in \bar{J}\}$. On peut supposer que $\delta_i = 1$, $L_i^* = L_j$ pour $i \in \bar{I}$, et $\delta_i = L_i^* = 0$ sinon.

De même, $U_{i,j} = 1$ si $i = \hat{i}(j)$, $j \in \bar{J}$, et $U_{i,j} = 0$ sinon.

Soit $\bar{I}^A = \{i \in \llbracket 0, N-1 \rrbracket : \bar{z}_i = 1\}$. \bar{I}^A peut être divisé en intervalles disjoints de la forme $[f_k, g_k]$, $k = 1, \dots, q$, sur lesquels la micro-usine est active. Notons que $\bar{z}_{f_k-1} = 0 = \bar{z}_{g_k+1}$, si $f_k > 0$, $g_k < N-1$, $1 \leq k \leq N-1$. Comme la micro-usine ne peut pas être active pendant que le véhicule fait le plein, $\bar{I} \cap \bar{I}^A = \emptyset$, ceci implique que (4.2b) est satisfait.

Définissons \bar{y}_i :

$$\bar{y}_i = \begin{cases} 1 & \text{si } i = f_k, \text{ pour un certain } k \leq q \\ 0 & \text{sinon.} \end{cases}$$

(4.2a)-(4.2c) sont donc vérifiés par \bar{y} et \bar{z} . A partir de (4.3a), l'application itérative de l'équation (4.3d) nous permet de déterminer le volume V_i^{Tank} , pour $i = 0, \dots, N - 1$. Toutes les autres inégalités sont des contraintes de capacité qui sont nécessairement satisfaites par une solution réalisable de SMEPC.

Réciproquement, considérons une solution optimale réalisable

$$(\bar{y}, \bar{\delta}, \bar{x}, \bar{L}, \bar{z}, \bar{L}^*, \bar{T}, \bar{T}^*, \bar{V}^{Veh}, \bar{V}^{Tank}, \bar{U})$$

de $MILP_{SMEPC}$.

Par les contraintes (4.6a) et (4.6b), le vecteur \bar{U} définit une correspondance $j \leftarrow \hat{i}(j)$ entre les ensembles $\bar{J} = \{j \in \{0, \dots, M\} : \bar{x}_j = 1\}$ et $\bar{I} = \{i \in \{0, \dots, N-1\} : \bar{\delta}_i = 1\}$. Cette correspondance est cohérente avec l'ordre linéaire standard : si $j_1 < j_2$ alors $\hat{i}(j_1) < \hat{i}(j_2)$. En effet, les inégalités ((4.5a)-(4.5d)) assurent que $T_{j+1} > T_j, \forall j$ et $T_{j+1} > T_j^* > T_j, \forall j \in \bar{J}$. Donc $T_{j_2}^* > T_{j_2} \geq T_{j_1+1} > T_{j_1}^*$. D'après (4.6c)-(4.6d), nous déduisons que $\hat{i}(j_2) \times p > T_{j_2} > T_{j_1} \geq \hat{i}(j_1) \times p$. Par conséquent, $\hat{i}(j_1) < \hat{i}(j_2)$. Par optimalité de la solution, les contraintes (4.5c) et (4.6c) sont fortes. Ainsi, \bar{T} définit les dates d'arrivée du véhicule. Comme $L_{\hat{i}(j)}^* = L_j, j \in \bar{J}$ d'après les contraintes linéarisées (4.6e), le réservoir du véhicule et de la micro-usine ont des volumes réalisables en raison des contraintes de flux et de capacité.

Nous concluons qu'une solution de SMEPC peut être extraite de la solution optimale réalisable de $MILP_{SMEPC}$, les deux ayant le même coût.

4.9.2 Démonstration du lemme 2

Cette section présente la démonstration du lemme 2. Si $RMILP_{SMEPC}$ admet une solution optimale

$$(\bar{y}, \bar{\delta}, \bar{x}, \bar{L}, \bar{z}, \bar{L}^*, \bar{T}, \bar{T}^*, \bar{V}^{Veh}, \bar{V}^{Tank}, \bar{U})$$

telle que $\bar{Z}_r = 0$. Alors

$$\bar{y}_i = \bar{z}_i = 0, \text{ pour } i = 0, \dots, N - 1.$$

Puis de la contrainte (4.3d), on obtient que $V_{i+1}^{Tank} = V_i^{Tank} - L_i^*$, pour $i = 0, \dots, N - 1$. Par les contraintes (4.3a) et (4.3b), $L_i^* = 0$, pour $i = 0, \dots, N - 1$. La première équation de la linéarisation de (4.6e) implique que $m_{i,j} = 0$, pour $j = 0, \dots, M, i = 0, \dots, N - 1$. Des deux dernières équations de la linéarisation de (4.6e), $L_j = 0$, pour $j = 0, \dots, M$. Dorénavant, (4.4e) donne $V_{j+1}^{Veh} < V_j^{Veh}$, pour $j = 0, \dots, M$. Mais ceci contredit les conditions initiales et finales (4.4a) et (4.4a).

4.9.3 Démonstration du théorème 3

Cette section présente la démonstration du théorème 3. Examinons le cas spécifique suivant de SMEPC :

- N est donné comme un nombre pair $N = 3.Q + 1$, où Q est un paramètre supposé être ≥ 2 ;
- $p = 1$;
- Le coefficient α de la fonction objectif est nulle ;
- Le coût d'activation de la micro-usine $Cost^F = 0$ est nul ;
- Les coûts de production dépendants du temps $Cost_i^V$ sont nuls dès que $i \geq Q$;
- $TMax = N = 3 \times Q + 1$;
- Les rendement de production R_i sont nuls dès que $i \geq Q$;

- La capacité maximale du véhicule C^{Veh} et celle de la citerne d'hydrogène C^{Tank} sont infinies ;
- La quantité d'hydrogène initiale dans la citerne est $H_0 = 0$
- La quantité d'hydrogène initiale dans le véhicule est $E_0 = 1$
- $M = 1$ et la tournée $\Gamma = \{Depot, 1, Depot\}$ et :
 - $d_1 = d_1^* = Q$; $d_0 = d_0^* = 1$;
 - $t_0 = t_1 = Q$;
 - $\varepsilon_1 = \varepsilon_1^* = (H - 1)/2$, où H est un paramètre supposé être ≥ 2 ;
 - $\varepsilon_0 = \varepsilon_0^* = 1$;
 - $e_0 = e_1 = (H - 1)/2$.

Ensuite, on voit que la micro-usine doit produire H unités d'hydrogène pendant les périodes $0, \dots, Q - 1$, de telle sorte que le véhicule puisse partir à la date $Q - 1$ et effectuer sa tournée en $2Q + 1$ périodes (y compris $p = 1$ unité de temps pour faire le plein, voir figure (4.14)) et revenir au dépôt final avec un réservoir chargé d'une unité d'énergie hydrogène. Il s'ensuit que dans ces hypothèses, minimiser le coût économique du processus signifie résoudre l'instance suivante du sac à dos :

Calculer un vecteur de booléens $Z = (Z_0, \dots, Z_{Q-1})$ tel que :

- $\sum_{i=0, \dots, Q-1} R_i \times Z_i \geq H$;
- $\text{Min} = \sum_{i=0, \dots, Q-1} Cost_i^V \times Z_i$.

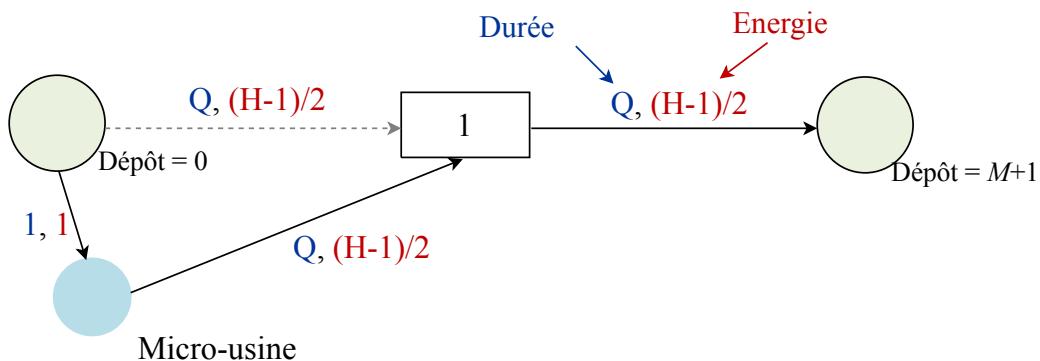


FIGURE 4.14 – Tournée du véhicule. Le véhicule se recharge à la micro-usine pendant $p = 1$ unité de temps.

CHAPITRE 5

SCHÉMA DE PROGRAMMATION DYNAMIQUE POUR RÉSOUTRE SMEPC

Sommaire

5.1 Introduction	117
5.2 Un algorithme de programmation dynamique pour SMEPC : DPS_SMEPC	117
5.2.1 Espace temps et états	117
Etat initial et Etats finaux	118
5.2.2 Variables de décision	119
5.2.3 Pré-conditions, transitions et coûts liés aux décisions et aux états	120
5.2.4 Mise en œuvre algorithmique	127
Equations de Bellman	127
Stratégie d'exploration	128
Algorithme général du DPS_SMEPC	128
5.3 Un schéma d'approximation polynomial (PTAS) : filtrage par arrondi (Rounding)	129
5.4 Mécanismes de filtrage	131
5.4.1 Mécanismes de filtrage par relations de dominance	132
5.4.2 Mécanismes de filtrage logique	132
5.4.3 Mécanismes de filtrage par estimation optimiste	134
Borne supérieure calculée par une adaptation gloutonne du DPS_SMEPC	137
5.4.4 Mécanismes de filtrage heuristique	137
5.5 Description d'une heuristique rapide <i>He</i> pour résoudre SMEPC	139
5.5.1 Le graphe	140
5.5.2 Les poids sur les arcs	140
5.5.3 Exemple	141
5.6 Expérimentations numériques	142
5.6.1 Objectifs et contexte technique	142
5.6.2 Instances	142
5.6.3 Recherche d'une borne supérieure de qualité pour le DPS_SMEPC	146
Recherche d'une borne supérieure de qualité pour le DPS_SMEPC : les instances du paquet INST_VAR	148
Recherche d'une borne supérieure de qualité pour le DPS_SMEPC : les instances du paquet INST_CTE	149

5.6.4 Impact des mécanismes de filtrage du DPS_SMEPC	150
Impact des mécanismes de filtrage du DPS_SMEPC : les instances du paquet	
INST_VAR	152
Impact des mécanismes de filtrage du DPS_SMEPC : les instances du paquet	
INST_CTE	154
5.7 Conclusion	155
5.8 Annexes	155
5.8.1 Démonstration du théorème 4	155

5.1 Introduction

Ce chapitre présente un schéma de programmation dynamique (DPS) pour résoudre le problème **SMEPC**, nous désignons par **DPS_SMEPC** ce DPS [24]. La **section 5.2** aborde l'architecture du DPS du problème **SMEPC** et les pseudo-codes de ce dernier. La **section 5.3** présente un schéma d'approximation polynomiale du problème **SMEPC**. La **section 5.4** liste les mécanismes de filtrage du schéma de programmation dynamique **DPS_SMEPC**. Ces mécanismes de filtrage permettront de diminuer le nombre d'états présents à un temps au sens de la programmation dynamique. On mesurera l'impact de ces mécanismes de filtrage sur le nombre d'états du **DPS_SMEPC**. La **section 5.6** présente les résultats expérimentaux réalisés sur le **DPS_SMEPC**. Pour conclure, la **dernière section** synthétise ce chapitre.

5.2 Un algorithme de programmation dynamique pour SMEPC : DPS_SMEPC

On présente le schéma de programmation dynamique nommé **DPS_SMEPC** conçu pour calculer la solution optimale d'une instance du problème **SMEPC**. On explique son architecture et sa mise en œuvre algorithmique. Pour présenter l'architecture du problème **SMEPC**, on présente d'abord l'espace temps au sens de la programmation dynamique et les variables qui forment un état au sens de la programmation dynamique. Ensuite, on liste l'ensemble des décisions qui peuvent être prises à partir d'un état connu. Enfin, on écrit tous les états qui peuvent résulter de chaque décision ainsi que les coûts de transition et les pré-conditions à respecter pour qu'un état soit généré.

5.2.1 Espace temps et états

L'espace temps de notre DPS est l'ensemble \mathcal{T} des couples (i, j) , $i = 0, \dots, N$, $j = 0, \dots, M + 1$ qui représentent le temps au sens de la programmation dynamique, fourni avec son ordre partiel standard. Nous pouvons étendre cet ordre partiel à un ordre linéaire de plusieurs façons, en associant à chaque couple $(i, j) \in \mathcal{T}$ (qui représente le temps au sens de la programmation dynamique) un successeur $Succ_{\Delta}(i, j)$. Par exemple, en fixant :

$$Succ_{\Delta}(i, j) = \begin{cases} (i + 1, j) & \text{si } i \leq N - 1. \\ (0, j + 1) & \text{sinon.} \end{cases}$$

Nous relierons les périodes i et les stations j en introduisant des relations ($<<$, $>>$, $==$) qui expriment le positionnement relatif d'une période i et la date T où le véhicule se trouve à la station j . C'est-à-dire que nous fixons pour tout $i \in \{0, \dots, N - 1\}$ et $T \in \{0, \dots, TMax\}$:

- $T << i$ si $T < p \times i$;
- $T >> i$ si $T \geq p \times (i + 1)$;
- $T == i$ si $p \times i \leq T < p \times (i + 1)$.

Ces relations vont nous aider à définir les variables d'état du **DPS_SMEPC**. Pour tout couple $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique, on associe un état défini par un quadruplet $s = (Z, T, V^{Tank}, V^{Veh})$, avec :

- Z : l'état de la micro-usine de production d'hydrogène ;

$$Z = \begin{cases} 1 & \text{si la micro-usine est active à la fin de la période } i - 1. \\ 0 & \text{sinon.} \end{cases}$$

- V^{Tank} : la quantité d'hydrogène dans la citerne à la micro-usine au début de la période i ;
- V^{Veh} : la quantité d'hydrogène dans le réservoir du véhicule lorsqu'il arrive à la station j ;
- $T \in 0, \dots, TMax$ est une date signifiant :
 - Si $T >> i$ alors le véhicule est en route vers la station j et il y arrivera à la date T (Voir figure (5.1));
 - Si $T << i$ alors le véhicule est entre la station j et la micro-usine. Le véhicule peut être entrain d'attendre à la micro-usine pour être rechargé (Voir figure (5.1));
 - Si $T == i$ alors le véhicule se trouve à la station j , et doit choisir entre aller à la station $j + 1$ ou aller à la micro-usine se recharger (Voir figure (5.1)).

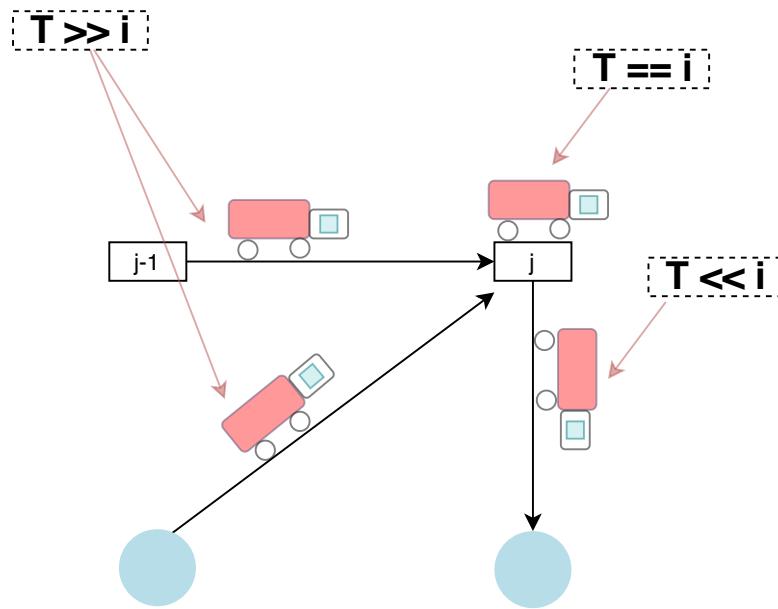


FIGURE 5.1 – Le véhicule parcourt 3 positionnements relatifs possibles en fonction de i et j : soit le véhicule est en route vers la station j ($i << T$), soit le véhicule est à la station j ($i == T$), soit le véhicule est entre la station j et la micro-usine ($i >> T$).

Remarque 3 Nous voyons que selon cette explication, le positionnement relatif de T et i par rapport aux relations $==$, $<<$ et $>>$ agit comme une variable d'état implicite, ce qui nous aidera à identifier les décisions qui peuvent être associées à un couple courant $(i, j) \in \mathcal{T}$ (qui représente le temps au sens de la programmation dynamique) et un état courant s , ainsi que leur impact.

On vient de donner la signification de chaque variable d'un état au sens de la programmation dynamique du **DPS_SMEPC**. De plus, on a décrit le temps au sens de la programmation dynamique du **DPS_SMEPC**. On va maintenant donner la signification de chaque variables de décisions.

Etat initial et Etats finaux

L'état initial correspond au couple $(0, 0)$ et au quadruplet $s_0 = (0, 0, H_0, E_0)$. Le couple $(0, 0)$ représente le temps initial au sens de la programmation dynamique. On commence la tournée à la date $i = 0$ et au dépôt initial ($j = 0$), avec une micro-usine qui ne produit pas de l'hydrogène ($Z = 0$). La date T d'arrivé à la station $j = 0$ est 0. H_0 est la quantité d'hydrogène initialement présente dans

la citerne de la micro-usine. E_0 est la quantité d'hydrogène initialement présente dans le réservoir du véhicule.

L'état final correspond au couple $(i \leq N, M + 1)$, qui représente le temps au sens de la programmation dynamique à l'état final, et à tout quadruplet $(Z, T \leq TMax, V^{Tank} \geq H_0, V^{Veh} \geq E_0)$ car la tournée finit au dépôt final $M + 1$. Lorsque le véhicule finit sa tournée, la quantité d'hydrogène dans le réservoir du véhicule et la quantité d'hydrogène dans la citerne de la micro-usine doivent respectivement être au moins égale à E_0 et H_0 . On veut ici éviter les stratégies triviales qui consistent à ne pas produire et à ne pas recharger tout au long de la tournée.

5.2.2 Variables de décision

Une décision D relative à un couple $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique et un état $s = (Z, T, V^{Tank}, V^{Veh})$ est un triplet $D = (z, x, \delta)$ dans $\{0, 1\}^3$, signifiant :

- $z = 1$ signifie que la micro-usine produit de l'hydrogène pendant la période i alors qu'elle ne produisait pas de l'hydrogène à la période $i - 1$;

$$z = \begin{cases} 1 & \text{si la micro-usine est active à la fin de la période } i \\ 0 & \text{sinon.} \end{cases}$$

- x désigne une décision prise dès que $T == i$:

$$x = \begin{cases} 1 & \text{si le véhicule se recharge en hydrogène à la micro-usine en se déplaçant de } j \text{ à } j + 1. \\ 0 & \text{s'il se déplace de la station } j \text{ à la station } j + 1 \text{ sans se recharger.} \end{cases}$$

- $\delta = 1$ signifie que le véhicule est à la micro-usine et décide de faire le plein de son réservoir d'hydrogène durant la période i . **On suppose que la recharge dure une période complète et en interdisant à la micro-usine d'être active pendant cette période.**

$$\delta = \begin{cases} 1 & \text{si le véhicule décide de se recharger durant la période } i \\ 0 & \text{sinon.} \end{cases}$$

Définition 3 Une variable de décision est neutralisée lorsqu'elle ne prend aucune valeur.

Les variables de décisions décrites ci-dessus nous permettront de lister par la suite l'ensemble des décisions possibles. On a $2^3 = 8$ décisions au total, parmi lesquels cinq sont réalisables. Les trois décisions non réalisables sont :

- $z = 0, x = 1, \delta = 1$: cette décision est interdite car lorsqu'on prend la décision de se recharger ($x = 1$) le véhicule ne se trouve pas encore à la micro-usine mais à la station j . Et pour se déplacer de j à la micro-usine, il faut au moins $d_j > 0$ temps au véhicule. Le véhicule ne se recharge pas à la période i parce que la recharge dure une période et le seul endroit où il se recharge c'est à la micro-usine ;
- $z = 1, x = 0/1, \delta = 1$: ces décisions sont interdites car on interdit que la production et la recharge se fassent simultanément pour éviter des pannes du système.

Pour chaque décision possible, on va lister ses conditions de faisabilité appelées ici **pré-conditions**.

5.2.3 Pré-conditions, transitions et coûts liés aux décisions et aux états

Une décision est prise à la fin d'une période $i - 1$. Nous allons maintenant considérer le couple $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique, un état courant $s = (Z, T, V^{Tank}, V^{Veh})$, et analyser l'ensemble de toutes les décisions possibles $D = (z, x, \delta)$ en fonction de la position du véhicule par rapport à la station j . Pour chacune de ces décisions D , nous allons détailler les conditions qui assureront la faisabilité de D , la transition qui en résulte $((i, j), s) \rightarrow ((i', j'), s')$ et le coût CT de cette transition. Les trois positions possibles du véhicule par rapport à la station j sont :

Pour le cas $T >> i$:

Le véhicule est en route vers la station j et il y arrivera à la date T (Voir figure (5.1)). Les variables de décisions x et δ sont neutralisées. On a deux décisions possibles : produire durant la période i ($z = 1$) ou ne pas produire durant la période i ($z = 0$). Plus précisément :

- 1) $z = 1$: la micro-usine produit de l'hydrogène pendant la période i . Pour que cette décision soit prise, la condition $V^{Tank} + R_i \leq C^{Tank}$ doit être vérifiée. C'est-à-dire qu'en produisant pendant la période i , la quantité d'hydrogène dans la citerne ne dépasse pas la capacité maximale de la citerne. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$ et le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V$.
- 2) $z = 0$: la micro-usine ne produit pas de l'hydrogène pendant la période i . Cette décision est toujours possible. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$ et le coût de transition est nul.

Pour le cas $T << i$:

le véhicule est entre la station j et la micro-usine (figure (5.1)) ou entrain d'attendre à la micro-usine pour se recharger. On a deux cas possible suivant que le véhicule arrive à la micro-usine avant ou au début de la période i (i.e. $T + d_j \leq p \times i$) ou après le début de la période i (i.e. $T + d_j > p \times i$).

1. **1^{er}cas** $T + d_j \leq p \times i$: le véhicule arrive à l'usine avant le début de la période i . La variable x est neutralisée. On a trois décisions possibles : produire durant la période i et faire attendre le véhicule à la micro-usine ($z = 1, \delta = 0$) ou ne pas produire durant la période i et faire attendre le véhicule à la micro-usine ($z = 0, \delta = 0$) ou ne pas produire durant la période i et recharger le véhicule durant la période i ($z = 0, \delta = 1$). Plus précisément :

- 1) $z = 1, \delta = 0$: la micro-usine produit de l'hydrogène pendant la période i et le véhicule attend à la micro-usine pour se recharger. Les pré-conditions suivantes doivent être vérifiées :

$$\square V^{Tank} + R_i \leq C^{Tank};$$

- $$\square p \times (i + 1) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax : \text{le véhicule peut attendre à la micro-usine s'il a encore suffisamment de temps pour se recharger à la période suivante (il sortira alors de l'usine à la date } p \times (i + 1) + p\text{), puis pour se déplacer de la micro-usine à la station } j + 1 \text{ (}} d_{j+1}^*\text{) et enfin pour finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer (}} \sum_{k \geq j+1} t_k\text{).}$$

Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$ et le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V$.

- 2) $z = 0, \delta = 0$: la micro-usine ne produit pas de l'hydrogène pendant la période i et le véhicule attend à la micro-usine pour se recharger. Pour que la décision $\delta = 0$ soit réalisable, le véhicule doit avoir suffisamment de temps pour attendre, c'est-à-dire $p \times (i + 1) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$ et le coût de transition est nul.
- 3) $z = 0, \delta = 1$: la micro-usine ne produit pas de l'hydrogène pendant la période i et le véhicule décide de se recharger à la période i . **On suppose que la recharge dure une période et en interdisant à la micro-usine d'être active pendant cette période.** Cela signifie que $T \ll i$ et que la différence $p.i - T$ est au moins égale au temps nécessaire au véhicule pour passer de j à la micro-usine car le véhicule doit être à la micro-usine pour pouvoir se recharger.

Soit $Tour_ = \varepsilon_{j+1}^* + \sum_{k \geq j+1} e_k + E_0$ la quantité d'hydrogène nécessaire au véhicule pour partir de la micro-usine et finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer après la station j . Soit Q_Rech la quantité d'hydrogène rechargée par le véhicule durant la période i .

La pré-condition suivante doit être vérifiée :

- $\varepsilon_{j+1} + \varepsilon_{j+1}^* \leq Inf(C^{Veh}, V^{Tank} + V^{Veh} - \varepsilon_j)$: la quantité d'hydrogène dans le réservoir du véhicule après la recharge doit être au moins égale à la quantité d'hydrogène nécessaire pour aller en $j + 1$ et revenir à la micro-usine.

Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j + 1)$. L'état résultant est $s' = (0, p \times (i + 1) + d_{j+1}^*, V^{Tank} - Q_Rech, V^{Veh} + Q_Rech - \varepsilon_j - \varepsilon_{j+1}^*)$. Le coût de transition est $\alpha \times (p \times (i + 1) + d_{j+1}^* - T)$.

Concernant la quantité rechargée par le véhicule durant la période i , on a trois possibilités :

- Soit le véhicule vide la citerne, ce qui signifie qu'il a suffisamment de place dans son réservoir pour stocker l'hydrogène contenu dans la citerne, auquel cas $Q_Rech = V^{Tank}$;
- Soit le véhicule remplit son réservoir, ce qui signifie que la citerne contient suffisamment d'hydrogène pour remplir le réservoir du véhicule, auquel cas $Q_Rech = C^{Veh} - V^{Veh} + \varepsilon_j$;
- Soit le véhicule recharge exactement ce dont il a besoin pour partir de la micro-usine et finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer après la station j , auquel cas $Q_Rech = Sup(0, Tour_ - V^{Veh} + \varepsilon_j)$.

Donc, la quantité rechargée est $Q_Rech = Inf(V^{Tank}, C^{Veh} - V^{Veh} + \varepsilon_j, Sup(0, Tour_ - V^{Veh} + \varepsilon_j))$ car on veut éviter de recharger trop d'énergie alors que le véhicule n'en a pas besoin pour finir la tournée.

2. **2^ecas** $T + d_j > p \times i$: le véhicule se déplace de la station j à la micro-usine et il ne peut pas encore se recharger pendant la période i car il arrive à la micro-usine à une date appartenant à la période $]i, i + 1[$

Les variables de décisions x et δ sont neutralisées. On a deux décisions possibles : produire durant la période i ($z = 1$) ou ne pas produire durant la période i ($z = 0$). Plus précisément :

- 1) $z = 1$: la micro-usine produit de l'hydrogène pendant la période i . La pré-condition $V^{Tank} + R_i \leq C^{Tank}$ doit être vérifiée. Concernant l'évolution du temps au sens de la

programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$ et le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V$.

- 2) $z = 0$: la micro-usine ne produit pas de l'hydrogène pendant la période i . Cette décision est toujours possible. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$ et le coût de transition est nul.

Pour le cas $T == i$:

Le véhicule est à la station j et il doit choisir entre aller à la station $j + 1$ ou aller à la micro-usine se recharger (figure (5.1)). Dans ce cas, on a trois ou quatre décisions possibles. En effet, on a 2 décisions sont possibles quand le véhicule décide d'aller se recharger ($x = 1$) : la micro-usine produit pendant la période i ou ne produit pas en i . Par contre, quand le véhicule décide de continuer sa tournée vers la station $j + 1$, le nombre de décisions dépendent de sa date d'arrivée à la station $j + 1$:

- Si le véhicule arrive à la station $j + 1$ pendant la période i alors la décision de production à la période i est **remise à plus tard**. z est donc neutralisé ;
- Si le véhicule arrive à la station $j + 1$ après la fin de la période i alors deux décisions sont possibles : la micro-usine produit pendant la période i ou ne produit pas en i .

1. $x = 1$: le véhicule décide de partir se recharger à la micro-usine ;

Pour ces deux décisions suivantes, la pré-condition suivante doit être satisfaite :

- $Sup(p \times (i + 1), T + d_j) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$: le véhicule va à la micro-usine s'il a encore suffisamment de temps pour se recharger, pour se déplacer de la micro-usine à la station $j + 1$ (d_{j+1}^*) et pour finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer ($\sum_{k \geq j+1} t_k$).

On a deux décisions possibles : produire durant la période i ($z = 1$) ou ne pas produire durant la période i ($z = 0$). Plus précisément :

- 1) $z = 1$: la micro-usine produit de l'hydrogène pendant la période i . La pré-condition $V^{Tank} + R_i \leq C^{Tank}$ doit être vérifiée ; Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$ et le coût de transition est $(Cost^F \times (1 - Z) + Cost_i^V)$;
- 2) $z = 0$: la micro-usine ne produit pas de l'hydrogène pendant la période i . Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j)$. L'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$. Le coût de transition est nul.

2. $x = 0$: Le véhicule décide de partir à la station $j + 1$;

Pour ces décisions suivantes, la pré-condition suivante doit être satisfaite :

- $V^{Veh} - e_j - \varepsilon_{j+1} \geq 0$: le véhicule doit avoir suffisamment d'hydrogène pour aller à la micro-usine après être allé en $j + 1$.
- 1) $T + t_j == i$: Le véhicule arrive à la station $j + 1$ à une date appartenant à la période $]i, i + 1[$, z est alors neutralisé (i.e. on ne prend pas de décision concernant la production en i , cette décision sera prise plus tard).
Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i, j + 1)$. L'état résultant est $s' = (Z, T + t_j, V^{Tank}, V^{Veh} - e_j)$ et le coût de transition est $\alpha \times t_j$;

2) $T + t_j >> i$: Le véhicule arrive à la station $j + 1$ après la période i . On a deux décisions possibles : produire durant la période i ($z = 1$) ou ne pas produire durant la période i ($z = 0$). Plus précisément :

- $z = 1$: la micro-usine produit de l'hydrogène pendant la période i . la pré-condition $V^{Tank} + R_i \leq C^{Tank}$ doit être vérifiée. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j + 1)$. L'état résultant est $s' = (1, T + t_j, V^{Tank} + R_i, V^{Veh} - e_j)$ et le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V + \alpha \times t_j$;
- $z = 0$: la micro-usine ne change pas son état précédent c'est-à-dire que z est neutralisé. Concernant l'évolution du temps au sens de la programmation dynamique on passe du couple (i, j) au couple $(i + 1, j + 1)$. L'état résultant est $s' = (Z, T + t_j, V^{Tank}, V^{Veh} - e_j)$ et le coût de transition est $\alpha \times t_j$.

On vient de lister les décisions possibles en fonction de la position relative d'une période et de la date T d'arrivée du véhicule à la station j . Dans un souci de clarté, on énumère maintenant l'ensemble de toutes les décisions possibles $D = (z, x, \delta)$. Pour chaque décision, on donnera les pré-conditions nécessaires pour appliquer la décision, les transitions (temps et états) et les coûts liés aux décisions et aux états résultants.

On considère le couple $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique et un état courant $s = (Z, T, V^{Tank}, V^{Veh})$, on va analyser l'ensemble de toutes les décisions possibles $D = (z, x, \delta)$. Pour chacune de ces décisions D , nous allons détailler les conditions qui assureront la faisabilité de D , la transition qui en résulte $((i, j), s) \rightarrow ((i', j'), s')$ et le coût CT de cette transition.

1. $z = 1, x = 0, \delta = 0$: la micro-usine produit pendant la période i alors que le véhicule continue à rouler comme auparavant ;

Pour ces décisions suivantes, les pré-conditions suivantes doivent être vérifiées :

- $V^{Tank} + R_i \leq C^{Tank}$: en produisant pendant la période i , la quantité d'hydrogène dans la citerne ne dépasse pas la capacité maximale de la citerne ;
- Si $T == i$ alors $V^{Veh} - e_j - \varepsilon_{j+1} \geq 0$ et $T + t_j >> i$: lorsque le véhicule va à la station $j + 1$, une fois qu'il y ait il doit avoir suffisamment d'hydrogène pour arriver à la micro-usine ;
- Si $T << i$ alors $p \times (i + 1) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$: le véhicule peut attendre à la micro-usine s'il a encore suffisamment de temps pour se recharger à la période suivante (il sortira alors de l'usine à la date $p \times (i + 1) + p$), puis pour se déplacer de la micro-usine à la station $j + 1$ (d_{j+1}^*) et enfin pour finir la tournée sans plus se recharger ($\sum_{k \geq j+1} t_k$).

Si $T >> i$ ou $T << i$ alors $x = 0$ ne se réfère pas à une véritable décision, x est neutralisé. On passe de (i, j) à $(i + 1, j)$ et l'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$. Le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V$.

Si $T == i$ alors le véhicule décide de se déplacer de j à $j + 1$. On passe de (i, j) à $(i + 1, j + 1)$ et l'état résultant est $s' = (1, T + t_j, V^{Tank} + R_i, V^{Veh} - e_j)$. Le coût de transition est $Cost^F \times (1 - Z) + Cost_i^V + \alpha \times t_j$.

2. $z = 1, x = 1, \delta = 0$: la micro-usine produit pendant la période i et le véhicule décide de passer de j à la micro-usine afin de faire le plein ;

Pour que cette décision soit prise, les pré-conditions suivantes doivent être vérifiées :

- $T == i$;
- $V^{Tank} + R_i \leq C^{Tank}$;
- $Sup(p \times (i+1), T + d_j) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$.

On passe de (i, j) à $(i+1, j)$ et l'état résultant est $s' = (1, T, V^{Tank} + R_i, V^{Veh})$. Le coût de transition est $(Cost^F \times (1 - Z) + Cost_i^V)$.

3. $z = 0, x = 0, \delta = 0$: la micro-usine ne produit pas pendant la période i et le véhicule continue son chemin ;

Pour ces décisions suivantes, les pré-conditions suivantes doivent être vérifiées :

- Si $T == i$ alors $x = 0$ signifie que le véhicule est suffisamment alimenté en hydrogène pour passer de j à $j+1$: $V^{Veh} - e_j - \varepsilon_{j+1} \geq 0$;
- Si $T + d_i \leq p \times i$ alors le véhicule a décidé auparavant de faire le plein entre j et $j+1$, et se trouve actuellement à la micro-usine. Pour que la décision $\delta = 0$ soit réalisable, le véhicule doit avoir suffisamment de temps pour attendre, c'est-à-dire $p \times (i+1) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$.

Si $T >> i$ ou $T << i$ alors on passe de (i, j) à $(i+1, j)$, l'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$ et le coût de transition est nul.

Si $T == i$ alors on passe de (i, j) à $(i+1, j+1)$, sauf dans le cas où $T + t_j == i$: dans ce dernier cas, la date i reste inchangé. Dans tous les cas, l'état résultant est $s' = (Z, T+t_j, V^{Tank}, V^{Veh} - e_j)$ et le coût de transition est $\alpha \times t_j$.

4. $z = 0, x = 1, \delta = 0$: la micro-usine ne produit pas pendant la période i et le véhicule se dirige vers la micro-usine pour se recharger en hydrogène ;

Pour que cette décision soit prise, les pré-conditions suivantes doivent être vérifiées :

- $T == i$;
- $Sup(p \times (i+1), T + d_j) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$.

On passe de (i, j) à $(i+1, j)$ et l'état résultant est $s' = (0, T, V^{Tank}, V^{Veh})$ et Le coût de transition est nul.

5. $z = 0, x = 0, \delta = 1$: la micro-usine ne produit pas pendant la période i et le véhicule décide de se recharger en hydrogène pendant la période i . Soit $Tour_ = \varepsilon_{j+1}^* + \sum_{k \geq j+1} e_k + E_0$ la quantité d'hydrogène nécessaire au véhicule pour partir de la micro-usine et finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer après la station j .

Pour que cette décision soit prise, les pré-conditions suivantes doivent être vérifiées :

- $T + d_j \leq p \times i$;
- $\varepsilon_{j+1} + \varepsilon_{j+1}^* \leq Inf(C^{Veh}, V^{Tank} + V^{Veh} - \varepsilon_j)$: la quantité d'hydrogène dans le réservoir du véhicule après recharge doit être au moins égale à la quantité d'hydrogène pour réaliser le trajet aller et retour de la micro-usine à la station $j+1$.

On passe de (i, j) à $(i+1, j+1)$, l'état résultant est $s' = (0, p \times (i+1) + d_{j+1}^*, V^{Tank} - Q_{Rech}, V^{Veh} + Q_{Rech} - \varepsilon_j - \varepsilon_{j+1}^*)$. Le coût de transition est $\alpha \times (p \times (i+1) + d_{j+1}^* - T)$.

Le tableau (5.1) récapitule l'ensemble des pré-conditions du schéma de programmation dynamique **DPS_SMEPC**.

Numéro	Pré-conditions	Interprétations
1	$V^{Tank} + R_i \leq C^{Tank}$	En produisant pendant la période i , la quantité d'hydrogène dans la citerne ne dépasse pas la capacité maximale de la citerne
2	$p \times (i + 1) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$	le véhicule peut attendre à la micro-usine s'il a encore suffisamment de temps pour se recharger à la période suivante (il sortira alors de l'usine à la date $p \times (i + 1) + p$), puis pour se déplacer de la micro-usine à la station $j + 1$ (d_{j+1}^*) et enfin pour finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer ($\sum_{k \geq j+1} t_k$). Si $j = M + 1$ alors cette contrainte devient $p \times (i + 1) \leq TMax$. On fait cela pour pouvoir continuer à prendre la décision de produire ou pas lorsque le véhicule est arrivé au dépôt final.
3	$\varepsilon_{j+1} + \varepsilon_{j+1}^* \leq Inf(C^{Veh}, V^{Tank} + V^{Veh} - \varepsilon_j)$	la quantité d'hydrogène dans le réservoir du véhicule après recharge doit être au moins égale à la quantité d'hydrogène pour réaliser le trajet aller et retour de la micro-usine à la station $j + 1$.
4	$Sup(p \times (i + 1), T + d_j) + p + d_{j+1}^* + \sum_{k \geq j+1} t_k \leq TMax$	le véhicule va à la micro-usine s'il a encore suffisamment de temps pour se recharger, pour se déplacer de la micro-usine à la station $j + 1$ (d_{j+1}^*) et pour finir la tournée en supposant qu'il n'a pas d'autres recharges à effectuer ($\sum_{k \geq j+1} t_k$). Si $j = M + 1$ alors cette contrainte devient $p \times (i + 1) \leq TMax$. On fait cela pour pouvoir continuer à prendre la décision de produire ou pas lorsque le véhicule est arrivé au dépôt final.
5	$T \gg i$	le véhicule est en route vers la station j et il y arrivera à la date T
6	$T \ll i$	le véhicule est entre la station j et la micro-usine. Le véhicule peut être entrain d'attendre à la micro-usine pour être rechargeé
7	$T == i$	le véhicule se trouve à la station j , et doit choisir entre aller à la station $j + 1$ ou aller à la micro-usine se recharger
8	$V^{Veh} - e_j - \varepsilon_{j+1} \geq 0$ et $T + t_j \gg i$	lorsque le véhicule va à la station $j + 1$, une fois qu'il y ait il doit avoir suffisamment d'hydrogène pour arriver à la micro-usine
9	$T + d_j \leq p \times i$	le véhicule arrive à la station $j + 1$ après la période i

TABLE 5.1 – Liste des pré-conditions du DPS_SMEPC.

Pour illustrer le fonctionnement du DPS_SMEPC, on revient à l'exemple de la section 4.2 et on décrit la manière dont les états et décisions évolueront selon la solution possible décrite dans la figure (5.2).

Tournée du véhicule :

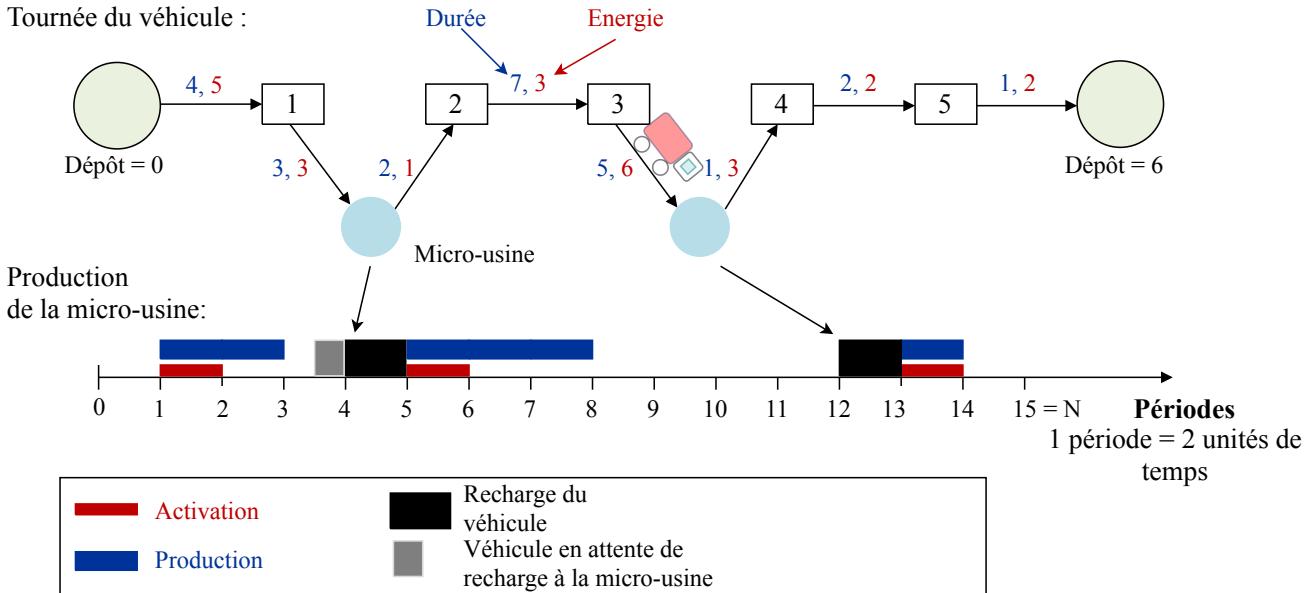


FIGURE 5.2 – Une solution faisable correspondant à $p = 2$, $E_0 = 8$, $H_0 = 4$, $TMax = 30$, $Cost^F = 7$, $C^{Tank} = 15$, $C^{Veh} = 15$, $\alpha = 1$, ainsi que t , d , e , ε comme aux figures (4.3) et (4.5).

Le tableau (5.2) nous donne ensuite l'évolution des couples $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique, les états $s = (Z, T, V^{Tank}, V^{Veh})$, les décisions $D = (z, x, \delta)$ et les coûts $Cost + \alpha \times T$, induits par cette solution.

temps DPS (i,j)		Etat $s = (Z, T, V^{Tank}, V^{Veh})$				Coût W	Décisions $D = (z, x, \delta)$		
i	j	Z	T	V^{Tank}	V^{Veh}	$W=Cost + T$	z	x	δ
0	0	0	0	4	8	0 + 0	0	0	0
1	1	0	4	4	3	0 + 4	1	0	0
2	1	1	4	9	3	8 + 4	1	1	0
3	1	1	4	13	3	9 + 4	0	0	0
4	1	0	4	13	3	9 + 4	0	0	1
5	2	0	12	0	12	9 + 12	1	0	0
6	3	1	19	3	9	18 + 19	1	0	0
7	3	1	19	8	9	20 + 19	1	0	0
8	3	1	19	12	9	22 + 19	0	0	0
9	3	0	19	12	9	22 + 19	0	1	0
10	3	0	19	12	9	22 + 19	0	0	0
11	3	0	19	12	9	22 + 19	0	0	0
12	3	0	19	12	9	22 + 19	0	0	1
13	4	0	27	0	12	22 + 27	1	0	0
14	5	1	29	4	10	30 + 29	0	0	0
15	6	0	30	4	8	30 + 30	*	*	*

TABLE 5.2 – Simulation de l'évolution du temps (i, j) , s , D et $Cost + \alpha \times T$ correspondant à la figure (5.2).

On a défini l'architecture du **DPS_SMEPC**, on présente dans la section suivante l'algorithme général. Pour cela, on définit l'état initial, les caractéristiques d'un état final et le principe de l'algorithme **DPS_SMEPC**.

5.2.4 Mise en œuvre algorithmique

Dans cette partie, on définit l'algorithme général du **DPS_SMEPC**, en donnant les données d'entrées, les sorties attendues, les équations de Bellman, et les instructions à exécuter.

Equations de Bellman

Pour tout couple (i, j) qui représente le temps au sens de la programmation dynamique et pour tout état $s = (Z, T, V^{Tank}, V^{Veh})$, nous associons la valeur W égale au coût le plus petit possible d'une séquence de transitions réalisables partant de l'état initial $s_0 = (0, 0, H_0, E_0)$ à la date $(0, 0)$ à l'état $s = (Z, T, V^{Tank}, V^{Veh})$ à la date (i, j) .

Nous concevons l'architecture **DPS_SMEPC** selon une stratégie orientée vers l'avenir voir figure (5.3). Pour tout couple $(i, j) \in \mathcal{T}$ qui représente le temps au sens de la programmation dynamique, nous désignons par $S(i, j)$ ce que nous appelons le sous-ensemble d'état lié à (i, j) et qui est en fait un ensemble de triplet $(s, W, Father)$ où W est la valeur définie ci-dessus et $Father$ signifie le couple $((i_f, j_f), s_f)$ où s_f est l'état lié au couple (i_f, j_f) qui nous a permis d'atteindre l'état s au couple $(i, j) \in \mathcal{T}$.

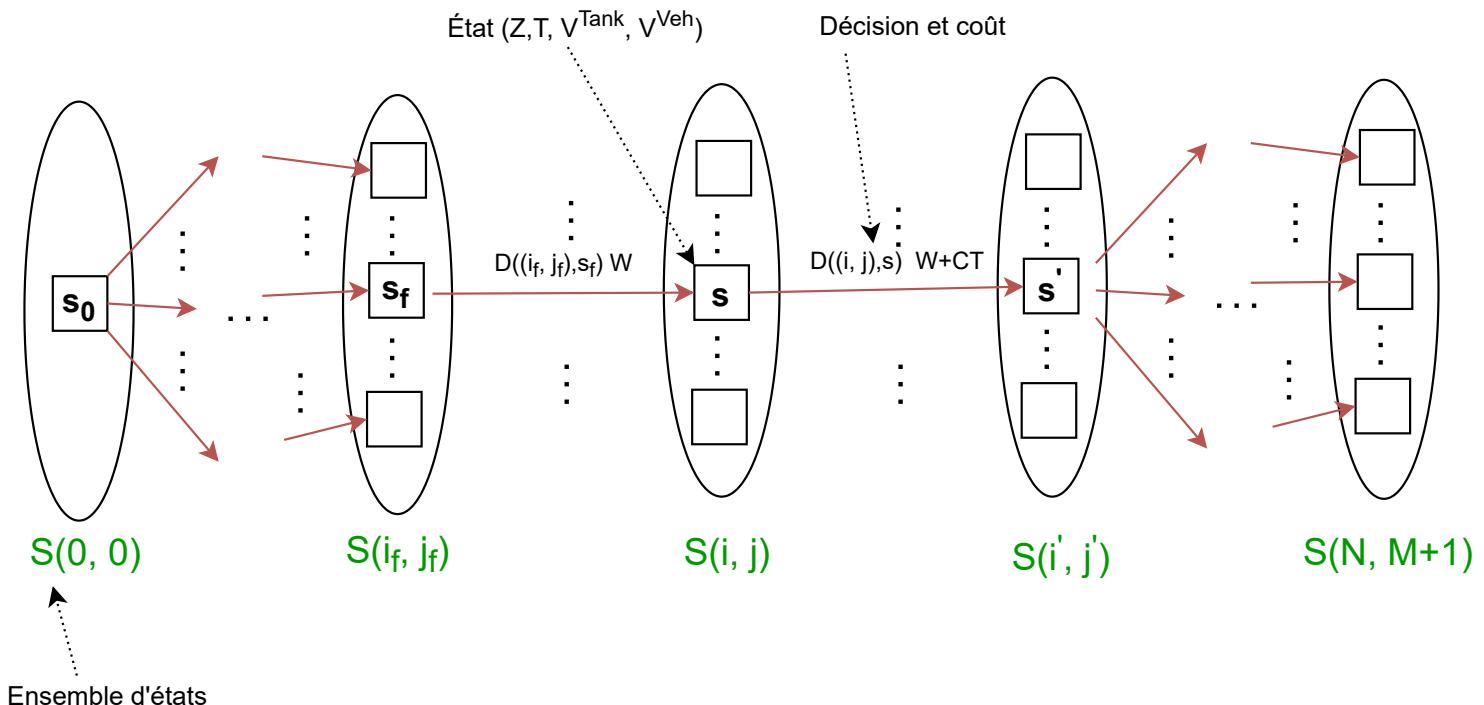


FIGURE 5.3 – La stratégie de génération des états du **DPS_SMEPC** est orientée vers l'avenir.

Nous analysons le sous-ensemble d'états $S(i, j)$, et pour tout état $s = (Z, T, V^{Tank}, V^{Veh})$, nous générerons l'ensemble de décisions possibles $Dec((i, j), s)$. Ensuite, pour chaque décision de ce type $D = (z, x, \delta)$, nous générerons le couple résultant (i', j') , qui représente le temps au sens de la programmation dynamique, et l'état $s' = (Z', T', V'^{Tank}, V'^{Veh})$, ainsi que la valeur $W + CT$, où CT signifie le coût de la transition entre $((i, j), s) \rightarrow ((i', j'), s')$ et W est la valeur associée à $(i, j) \in \mathcal{T}$ et s . On a trois scénarios possibles lors de l'ajout de s' dans $S(i', j')$:

- Si l'état s' n'apparaît pas encore dans l'ensemble d'états $S(i', j')$ alors on l'insère dans $S(i', j')$ avec la valeur $W + CT$;

- Si l'état s' apparaît déjà dans $S(i', j')$ avec une valeur $W^* > W + CT$, alors $W + CT$ devient la valeur associée à (i', j') et s' ;
- Si l'état s' apparaît déjà dans $S(i', j')$ avec une valeur $W^* \leq W + CT$, alors nous rejetons s' .

Stratégie d'exploration

La stratégie d'exploration du temps au sens de la programmation dynamique est la stratégie **Forward**.

Algorithme général du DPS_SMEPC

La description complète de l'algorithme général du DPS_SMEPC est résumée à l'algorithme (2).

Comme nous le verrons dans la section 5.4, l'instruction (11) doit être renforcée par des mécanismes de filtrage, qui vont nous permettre de tuer $(s', W', Father')$ au cas où une certaine forme d'in-faisabilité serait prévue, ou au cas où nous pourrions vérifier que le maintien de la tournée définie par (i', j') et $(s', W', Father')$ ne donnera pas une meilleure solution que celle qui existe déjà.

L'instruction (19) nous explique comment nous balayons l'espace-temps, d'une manière qui est cohérente avec son ordre partiel standard. Plusieurs fonctions $Succ_{\Delta}$ peuvent être définie, consistant à balayer Δ selon les lignes i , les colonnes j , ou les couches diagonales.

Algorithme 2 DPS_SMEPC

Entrées: $N, M, TMax, H_0, E_0, C^{Tank}, C^{Veh}$,

Sorties: $Current_Sol, Current_Value$

Initialisation :

- 1: $i \leftarrow 0; j \leftarrow 0; W \leftarrow 0;$
- 2: $S(0, 0) \leftarrow \{(s_0 = (0, 0, H_0, E_0), W = 0, Father = Indfini)\};$
- 3: $Current_Sol \leftarrow Undefined; Current_Value \leftarrow +\infty;$

Boucle principale :

- 4: **Tant que** $j + i \leq N + M + 1$ **faire**
 - 5: **pour** $(s, W, Father)$ dans la liste $S(i, j)$ **faire**
 - 6: Générer l'ensemble des décisions possibles $Dec((i, j), s)$ (au sens de la section 5.2.3);
 - 7: **pour** $D = (z, x, \delta)$ dans $Dec((i, j), s)$ **faire**
 - 8: Calculer le couple résultant (i', j') et l'état $s' = (Z', T', V^{Tank}, V^{Veh})$, ainsi que le coût de transition CT ;
 - 9: **Si** un triplet $(s', W', Father')$ apparaît déjà dans $S(i', j')$ et si $W' > W + CT$ **alors**
 - 10: Remplacer $(s', W', Father')$ par $(s', W + CT, (s, i, j))$;
 - 11: **sinon**
 - 12: Insérer $(s', W + CT, (s, i, j))$ dans $S(i', j')$;
 - 13: **Fin si**
 - 14: **Si** $(j' = M+1) \wedge (T' \leq TMax) \wedge (V^{Tank} \geq H_0) \wedge (V^{Veh} \geq E_0) \wedge (Current_Value > W + CT)$ **alors**
 - 15: $Current_Value \leftarrow W + CT;$
 - 16: $Current_Sol \leftarrow (s', (s, i, j));$
 - 17: **Fin si**
 - 18: **fin pour**
 - 19: **fin pour**
 - 20: $(i, j) \leftarrow Succ_{\Delta}(i, j);$
 - 21: **Fin tant que**
-

En utilisant l'algorithme global qu'on vient de présenter, on va donner un schéma d'approximation polynomial du problème **SMEPC**.

5.3 Un schéma d'approximation polynomial (PTAS) : filtrage par arrondi (Rounding)

Le **DPS_SMEPC** peut être en difficulté dès que M et N sont grands : chaque état est un quadruplet, avec T , V^{Veh} et V^{Tank} aux valeurs potentiellement importantes. Néanmoins, nous pouvons remarquer que si nous considérons $TMax$, C^{Tank} et C^{Veh} comme étant limités par des fonctions polynomiales de N et M , alors le **DPS_SMEPC** devient polynomiale dans le temps puisque le nombre d'états qu'il implique devient également limité par une fonction polynomiale de N et M .

En fait, nous pouvons aller plus loin et introduire un schéma d'arrondi du type schéma d'approximation polynomial (PTAS) comme suit :

1. **Arrondi d'états dans le schéma DPS_SMEPC** : L étant un nombre entier, alors, pour tout nombre entier n avec comme décomposition binaire $n = a_0 + a_1 \times 2 + \cdots + a_q \times 2^q$, nous nous sommes fixés :

- Si $q \leq L$ alors $Round(n, L) = n$ alors $Round(n, L) = a_{q-K} \times 2^{q-L} + \cdots + a_q \times 2^q$;
- Si $q \leq L$ alors $Round^*(n, L) = n$ alors $Round^*(n, L) = a_{q-K} \times 2^{q-L} + \cdots + a_q \times 2^q + 2^{q-L}$.

Deux nombres entiers n et m sont équivalents modulo les L plus grands bits si $Round(n, L) = Round(m, L)$.

Revenons maintenant à notre schéma algorithmique **DPS_SMEPC**. Nous supposons tout d'abord que toutes les entrées d , t , e , ε , R_i , $Cost^F$, $Cost_i^V$, $TMax$, C^{Tank} et C^{Veh} de notre modèle **SMEPC** sont des entiers. L étant un nombre entier, $s_1 = (Z, T_1, V_1^{Tank}, V_1^{Veh})$ et $s_2 = (Z, T_2, V_2^{Tank}, V_2^{Veh})$ étant deux états de notre schéma algorithmique **DPS_SMEPC**, par extension ils sont équivalents modulo les L plus grands bits si $Round(T_1, L) = Round(T_2, L)$, $Round(V_1^{Tank}, L) = Round(V_2^{Tank}, L)$, $Round(V_1^{Veh}, L) = Round(V_2^{Veh}, L)$.

2. **Le schéma d'approximation DPS_SMEPC(K)** :

Soit un certain entier $K \geq 1$, nous transformons l'algorithme (2) **DPS_SMEPC** en un algorithme paramétré **DPS_SMEPC(K)** en procédant comme suit :

- Nous calculons $L(N, M)$, avec une valeur minimale telle que $2^{L(N, M)} \geq N + M + 1$;
- Nous étendons la notion d'états, en considérant que tout état s est un quintuplet $(Z, \Omega, T, V^{Tank}, V^{Veh})$ associé à un couple (i, j) qui représente le temps au sens de la programmation dynamique, où Ω nous dit si $T << i$ ($\Omega = 0$), $T == i$ ($\Omega = 1$), $(T >> i) \wedge (T + d_j > p \times i)$ ($\Omega = 2$), $(T >> i) \wedge (T + d_j \leq p \times i)$ ($\Omega = 3$) : Cela signifie clairement que nous voulons tenir compte ici du fait (voir la remarque 3) que le positionnement relatif de T et i par les relations $<<$, $>>$ et $==$ agit comme une variable d'état implicite. En raison des effets d'arrondi, qui sont susceptibles de perturber ces relations, nous introduisons une variable Ω , dont le but est d'expliciter les informations fournies par ces positionnements relatifs ;
- Nous faisons en sorte que, à toute itération de la boucle principale de notre algorithme, l'ensemble $S(i, j)$ ne contienne pas deux triplets $(s_1, W_1, Father_1)$ et $(s_2, W_2, Father_2)$ de sorte que respectivement s_1 et s_2 , ainsi que W_1 et W_2 , soient équivalents modulo

les $K + 2.L(N, M)$ plus grands bits ; Nous le faisons en donnant la priorité aux triplets $(s_q, W_q, Father_q)$, $q = 1, 2$, lié à la plus petite valeur de W_q ;

- Nous remplaçons les valeurs initiales H_0 et E_0 par respectivement $Round^*(H_0, K + 1)$ et $Round^*(E_0, K + 1)$;
- Nous assouplissons la contrainte de capacité temporelle en remplaçant $TMax$ par $Round^*(TMax, K)$; De la même manière, nous assouplissons les contraintes de capacité d'hydrogène associées à la fois à la micro-usine et au véhicule en remplaçant les capacités C^{Tank} et C^{Veh} respectivement par $Round^*(C^{Tank}, K)$ et $Round^*(C^{Veh}, K)$: Cela signifie que $(i, j) \in \mathcal{T}$ étant le couple qui représente le temps au sens de la programmation dynamique actuelle, et s étant l'état actuel, nous calculons l'ensemble de décisions $Dec((i, j), s)$ (instruction 7 ci-dessous) de manière à ce que la faisabilité de toute décision $D = (z, x, \delta)$ soit vérifiée par rapport à la capacité temporelle $Round^*(TMax, K)$ et que les contraintes de capacité d'hydrogène soient vérifiées par rapport aux capacités d'hydrogène $Round^*(C^{Tank}, K)$ et $Round^*(C^{Veh}, K)$.

Algorithme 3 DPS_SMEPC(K)

Entrées: $N, M, TMax, H_0, E_0, C^{Tank}, C^{Veh}, K$

Sorties: $Current_Sol, Current_Value$

Initialisation :

- 1: $L \leftarrow L(N, M); i \leftarrow 0; j \leftarrow 0; W \leftarrow 0; Current_Sol \leftarrow Undefined;$
- 2: $Current_Value \leftarrow +\infty; S(0, 0) \leftarrow \{(s_0 = (0, 0, H_0, E_0), W = 0, Father = Indfini)\};$
- 3: $TMax \leftarrow Round^*(TMax, K); C^{Tank} \leftarrow Round^*(C^{Tank}, K);$
- 4: $C^{Veh} \leftarrow Round^*(C^{Veh}, K); H_0 \leftarrow Round^*(H_0, K + 1); E_0 \leftarrow Round^*(E_0, K + 1);$

Boucle principale :

- 5: **Tant que** $j + i \leq N + M + 1$ **faire**
- 6: **pour** $(s, W, Father)$ dans la liste $S(i, j)$ **faire**
- 7: Générer l'ensemble des décisions possibles $Dec((i, j), s)$ (au sens de la section 5.2.3) ;
- 8: **pour** $D = (z, x, \delta)$ dans $Dec((i, j), s)$ **faire**
- 9: Calculer le couple résultant (i', j') et l'état $s' = (Z', T', V'^{Tank}, V'^{Veh})$, ainsi que le coût de transition CT ;
- 10: **Si** un triplet $(s'', W'', Father'')$ apparaît déjà dans $S(i', j')$ tel que respectivement s' et s'' ainsi que $W + CT$ et W'' sont équivalents modulo les $K + L$ plus grande bits et $W + CT < W''$ **alors**
- 11: Remplacer $(s'', W'', Father'')$ par $(s', W + CT, (s, i, j))$;
- 12: **sinon**
- 13: Insérer $(s', W + CT, (s, i, j))$ dans $S(i', j')$;
- 14: **Fin si**
- 15: **Si** $(j' = M+1) \wedge (T' \leq TMax) \wedge (V'^{Tank} \geq H_0) \wedge (V'^{Veh} \geq E_0) \wedge (Current_Value > W + CT)$ **alors**
- 16: $Current_Value \leftarrow W + CT;$
- 17: $Current_Sol \leftarrow (s', (s, i, j));$
- 18: **Fin si**
- 19: **fin pour**
- 20: **fin pour**
- 21: $(i, j) \leftarrow Succ_{\Delta}(i, j);$
- 22: **Fin tant que**

Le DPS-SMPEPC(K) est résumé à l'algorithme (3).

Théorème 4 (Schéma d'approximation polynomial) K étant fixé, le **DPS_SMEPC(K)** est polynomial en temps. En outre, pour toute valeur $\varepsilon > 0$, on peut choisir K suffisamment grand de telle sorte que si le **SMEPC** admet une solution optimale avec la valeur W^{Opt} , alors **DPS_SMEPC(K)** donne une solution qui est réalisable en ce qui concerne les valeurs initiales $(1 + \varepsilon/2) \times H_0$ et $(1 + \varepsilon/2) \times E_0$, les valeurs seuils $(1 + \varepsilon) \times C^{Tank}$, $(1 + \varepsilon) \times C^{Veh}$ et $(1 + \varepsilon) \times TMax$ et dont la valeur de coût **Current_Value** n'est pas supérieure à $W^{Opt} + \varepsilon$.

La démonstration de ce théorème se trouve en annexe à la section 5.8.1.

Malgré le résultat ci-dessus, nous sommes confrontés à un problème qui est l'augmentation du nombre d'états lorsque N et M deviennent de plus en plus grand, et nous devons donc réfléchir à des dispositifs à mettre en place pour diminuer ces états. Dans la section suivante, on présentera quelques mécanismes de filtrage pour résoudre le problème d'explosion d'états.

5.4 Mécanismes de filtrage

Les mécanismes de filtrage permettent de diminuer le nombre d'états générés à chaque pas de temps au sens de la programmation dynamique. Si un état ne respecte pas les mécanismes de filtrage alors on le **tue** (i.e. on suppose qu'il n'existe aucune décision possible partant de cet état, l'état n'a donc pas d'état suivant). Dans la figure (5.4), au temps (i, j) au sens de la programmation dynamique, on a 12 états avant l'application des mécanismes de filtrage, une fois les mécanismes de filtrage appliquées, on remarque qu'il reste 6 états. C'est avec ses 6 états qu'on va continuer le processus de prise de décisions.

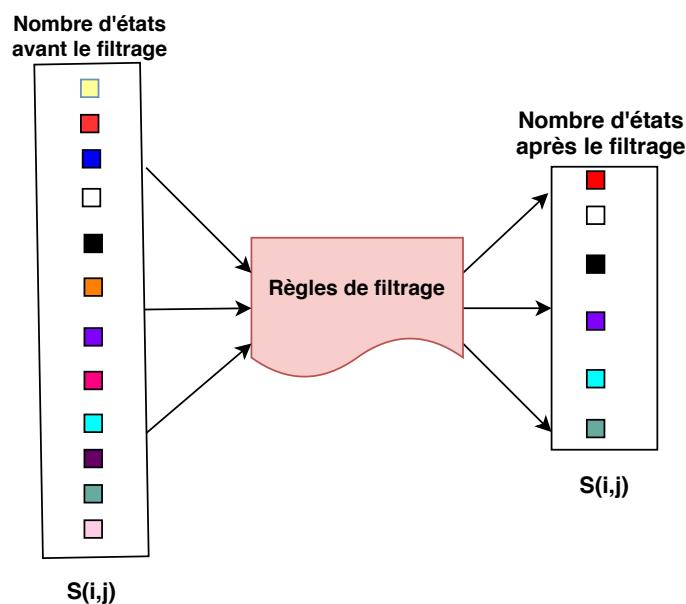


FIGURE 5.4 – Impact des mécanismes de filtrage sur le **DPS_SMEPC** à chaque pas de temps au sens de la programmation dynamique.

Les mécanismes de filtrage nous permettent de diminuer le nombre d'états générés par le **DPS_SMEPC**. Dans cette partie, on conçoit trois grands types de mécanismes de filtrage : les mécanismes de filtrage par relations de dominance, les mécanismes de filtrage logique et les mécanismes de filtrage basées sur la borne supérieure/inférieure. Les mécanismes de filtrage par relations de dominance consiste à

se débarrasser de tous les états dominés. Les mécanismes de filtrage logique sont basées sur l'anticipation des incohérences. Plus précisément, il s'agit de supprimer tous les états qui nous « conduirons » à des solutions non réalisables. Les mécanismes de filtrage basées sur la borne supérieure/inférieure consistent à supprimer tous les états qui nous « conduirons » à des solutions moins bonnes qu'une solution connue qui a été calculée avec une heuristique rapide.

5.4.1 Mécanismes de filtrage par relations de dominance

Les mécanismes de filtrage par relations de dominance consistent à supprimer des états qui sont dominés par d'autres états qui existent. Il existe deux types de mécanismes de filtrage par relations de dominance : les mécanismes de filtrage par relations de dominance forte et les mécanismes de filtrage par relations de dominance faible. Soit (i, j) un couple qui représente le temps au sens de la programmation dynamique du **DPS_SMEPC**. Si les deux états $s_1 = (Z_1, T_1, V_1^{Tank}, V_1)^{Veh}$ et $s_2 = (Z_2, T_2, V_2^{Tank}, V_2)^{Veh}$ donné respectivement avec les valeurs W_1 et W_2 et appartenant à (i, j) sont tels que : $W_1 \leq W_2, T_1 \leq T_2, Z_1 \geq Z_2, V_1^{Tank} \geq V_2^{Tank}, V_1^{Veh} \geq V_2^{Veh}$ alors s_1 domine s_2 , et nous tuons l'état s_2 (i.e. nous le retirons de la liste des états associés à (i, j)). Ce mécanisme de filtrage par relations de dominance forte a peu de pouvoir de filtrage, car elle est trop restrictive.

Néanmoins, d'autres mécanismes de filtrage par relation de dominance peuvent être mis en œuvre, par exemple le mécanisme de filtrage par relations de dominance faible heuristique. L'idée ici est de concevoir un mécanisme de filtrage flexible qui nous aidera à maintenir le nombre d'états sous un certain seuil noté NS . Soit A, B, C, D, E des paramètres positifs. Nous procédons de manière à ce que pour tout couple de temps (i, j) , nous ne gardions que les NS états ayant le plus petit : $A \times W + B \times T - C \times Z - D \times V^{Tank} - E \times V^{Veh}$. Le choix des valeurs (A, B, C, D, E) est un problème en soi. On peut par exemple considérer les valeurs de conversion suivantes : $(A, B, C, D, E) = (1, , Cost^F, Cost^V, Cost^V)$, où $Cost^V$ est la moyenne de $Cost_i^V, i = 0, \dots, N - 1$, valeurs.

5.4.2 Mécanismes de filtrage logique

Les mécanismes de filtrage logique tuent des états si ceux-ci ne respectent pas certains mécanismes logiques. Par exemple, s'il ne reste pas assez de temps ou d'énergie pour que le véhicule puisse finir sa tournée.

Considérons l'état $s = (Z, T, V^{Tank}, V^{Veh})$ appartenant au temps (i, j) au sens de la programmation dynamique, l'idée est d'anticiper l'in-faisabilité d'une tournée à partir des états s appartenant au temps (i, j) . Cette in-faisabilité peut être liée soit à un manque de temps (impossible de réaliser la tournée du véhicule avant l'échéance $TMax$), soit à la production d'énergie (impossible d'obtenir suffisamment d'hydrogène pour réaliser la tournée). Plus précisément, pour tout $j = 0, \dots, M$, nous obtenons une estimation approximative de l'énergie et du temps nécessaires pour permettre au véhicule d'aller de la station j au dépôt final en procédant comme suit :

- Nous désignons par $\Omega_0, \dots, \Omega_{M-j}$ les quantités $(\varepsilon_k + \varepsilon_{k+1}^* - e_k)$, $j \leq k \leq M$, ordonnées par ordre croissant $\Omega_0 < \dots < \Omega_{M-j}$;
- Nous désignons par $\Delta_0, \dots, \Delta_{M-j}$ les quantités $(d_k + d_{k+1}^* - t_k)$, $j \leq k \leq M$, ordonnées par ordre croissant. Pour toute station j
- Nous désignons par $Prod_Max(i)$ la quantité $\sum_{k \geq i} R_k$: la micro-usine ne peut produire plus de $Prod_Max(i)$ quantité d'hydrogène à partir de la période $p \times i$;

- Pour calculer H_j , $j = \{0, \dots, M\}$ l'énergie nécessaire au véhicule pour aller de la station j au dépôt final on exécute l'algorithme (4) nommé **Estimation-énergie(j)**. Le véhicule devra recharger au moins $H_j - V^{Veh}$ afin de réaliser sa tournée à partir de j et effectuer au moins $Refuel$ opérations de recharge en hydrogène. Pour calculer $Refuel_j$, $j = \{0, \dots, M\}$ le nombre de recharge nécessaire au véhicule pour aller de la station j au dépôt final on exécute l'algorithme (4) nommé **Estimation-énergie(j)**.
- En tenant compte du nombre de recharge $Refuel_j$, une borne inférieure noté $\Phi_j = \sum_{k \geq j} t_k + \sum_{q=0, \dots, Refuel_j-1} \Delta_q$ sur le temps qu'il faut au véhicule pour aller de j au dépôt final peut être déduite. Le véhicule a donc besoin d'au moins Φ_j unités de temps pour effectuer sa tournée de j au dépôt final ;

Algorithme 4 Estimation-énergie(j)

Entrées: $e, E_0, \Omega, C^{Veh}, j$

Sorties: $H, Refuel$

Initialisation :

- 1: $H \leftarrow E_0 + \sum_{k \geq j} e_k;$
- 2: $NotStop;$
- 3: $Refuel \leftarrow 0;$

Boucle principale :

- 4: **Tant que** $NotStop$ faire
 - 5: $Refuel_Aux \leftarrow \lfloor H/C^{Veh} \rfloor$
 - 6: **Si** $Refuel_Aux = Refuel$ alors
 - 7: $Stop;$
 - 8: **sinon**
 - 9: $H \leftarrow H + \sum_{q=Refuel, \dots, Refuel_Aux} \Omega_q;$
 - 10: $Refuel \leftarrow Refuel_Aux;$
 - 11: **Fin si**
 - 12: **Fin tant que**
-

Cela nous permet d'énoncer les mécanismes de filtrage logique suivantes :

1. Le mécanisme de filtrage basée sur le temps est le suivante si $(\Phi_j \geq TMax - T + 1)$, alors on tue l'état $s = (Z, T, V^{Tank}, V^{Veh})$ lié au temps (i, j) , au sens de la programmation dynamique, car il ne reste pas suffisamment de temps au véhicule pour finir sa tournée ;
2. Le mécanisme de filtrage basée sur l'énergie est énoncée comme suit si $H_j > V^{Veh} + Prod_Max(i) + V^{Tank}$, alors on tue l'état $s = (Z, T, V^{Tank}, V^{Veh})$ lié au temps (i, j) , au sens de la programmation dynamique, car il n'y aura pas suffisamment d'énergie pour que le véhicule puisse finir sa tournée.

Si nous nous référons à la description de l'algorithme (2) nommé **DPS_SMEPC**, nous voyons que l'instruction 12 doit ensuite être transformé en l'instruction suivante : appliquer successivement à s' le mécanisme de filtrage par relation de dominance forte, le mécanisme de filtrage basée sur le temps et le mécanisme de filtrage basée sur l'énergie ; Si s' n'a pas été tué alors insérer $(s', W + CT, (s, W, Father))$ dans $S(i', j')$.

5.4.3 Mécanismes de filtrage par estimation optimiste

Nous définissons, pour toute quantité d'énergie V et toute période i , le coût minimal de production $CostMin_{i,V,Z}$ requis pour produire V unités d'hydrogène entre la date $p \times i$ et la date $TMax$, Z désignant l'état de la micro-usine à la fin de la période $i - 1$: si $V = 0$ alors $CostMin_{N,V,Z} = 0$, sinon, $CostMin_{N,V,Z}$ n'est pas défini et, $CostMin_{i,V,Z} = Inf[CostMin_{i+1,V,0}, CostMin_{i+1,V-R_i,1} + (Cost^F \cdot (1 - Z) + Cost_i^V)]$.

Nous calculons $CostMin$ au moyen d'un DPS backward, et gardons en mémoire les valeurs $CostMin_{i,V,Z}$, à condition que la portée des valeurs de V ne soit pas trop grande (sinon nous effectuons un certain arrondissement des valeurs de V).

Si l'on donne maintenant une paire de temps (i, j) et un état connexe $s = (Z, T, V^{Tank}, V^{Veh})$ avec la valeur W . On obtient alors une borne supérieure/inférieure LB de la meilleure valeur **SMEPC** qui peut être dérivée de (i, j) et s : $LB((i, j), s) = \alpha \times \Phi_j + CostMin(i, (H_j - V^{Tank})^+, Z)$.

Si nous supposons maintenant que nous disposons d'une valeur faisable *Current_Value* de **SMEPC** réalisable, alors nous pouvons mettre en œuvre le mécanisme de filtrage suivante :

le filtrage basée sur la borne supérieure/inférieure est si $W + LB((i, j), s) \geq Current_Value$, alors on tue l'état $s = (Z, T, V^{Tank}, V^{Veh})$ associé au temps (i, j) .

Si nous nous référons à la description de l'algorithme **DPS_SMEPC** (2), nous voyons que l'instruction 12 doit ensuite être transformé en l'instruction suivante : appliquer successivement à s' le mécanisme de filtrage par relation de dominance forte, le mécanisme de filtrage basée sur le temps et le mécanisme de filtrage basée sur l'énergie ; Si s' n'a pas été tué alors appliquer à s' le mécanisme de filtrage basée sur la borne supérieure/inférieure ; Si s' n'a pas été tué alors insérer $(s', W + CT, (s, W, Father))$ dans $S(i', j')$.

Dans cette partie, on a défini les mécanismes de filtrage basées sur la borne supérieure/inférieure. Partant d'un état $s = (Z, T, V^{Tank}, V^{Veh})$ lié au temps (i, j) et dont la valeur est W , on a estimé le coût minimal (constitué du coût de production $CostMin_{i,V^{Veh},Z}$ et de la longueur de la tournée Φ_j) qu'il faut au véhicule pour aller de la station j au dépôt final. Si la valeur $W + CostMin_{i,(H_j - V^{Tank})^+,Z} + \alpha \times \Phi_j$ est plus grande que le coût d'une solution connue *Current_Value* (pré-calculée par exemple avec une heuristique) alors on tue l'état s .

Plus précisément, on va présenté par la suite comment est calculé $CostMin(i, V, Z)$ à l'aide d'un algorithme qu'on nommera « Cout minimal production ». Pour cela on présentera le principe, les données d'entrées, les sorties et le pseudo-code de l'algorithme « Cout minimal production ».

Principe de « Cout minimal production »

On calcule le coût minimal de production d'au moins V unités d'hydrogène entre l'instant $p \times i$ et l'instant $TMax$ à l'aide d'un programme dynamique qui fonctionne en backward. On suppose que l'état de la micro-usine pendant la période $i - 1$ est Z . On initialise $CostMin_{N,Z,0}$ à 0 car le coût de production de 0 unité d'hydrogène est nul, et $CostMin_{N,Z,V}$ est initialisée à $+\infty$.

A l'instant $p \times i$ on a deux cas possibles :

1. **On produit durant la période i la quantité R_i d'hydrogène**, il reste la quantité $V - R_i$ à produire à partir de la période $i + 1$. Pour calculer le coût de production, nous avons deux possibilités :

- Il faut démarrer la micro-usine c'est-à-dire que la micro-usine ne produisait pas pendant la période $i - 1$ ($Z = 0$) et on décide de produire pendant la période i . Le coût de production durant la période i est donc $Cost_i^v + Cost^F$. Il reste à calculer le coût minimal de production

d'au moins $V - R_i$ à partir de la période $i + 1$ sachant que la machine de production produit durant la période i ;

- La micro-usine est déjà en marche pendant la période $i - 1$ ($Z = 1$) et on décide de produire durant la période i . Le coût de production durant la période i est donc $Cost_i^v$. Il reste à calculer le coût minimal de production d'au moins $V - R_i$ à partir de la période $i + 1$ sachant que la machine de produit durant la période i .

2. **On ne produit pas durant la période i** donc je dois produire au moins la quantité V à partir de $i + 1$. Le coût de production durant la période i est 0. Il reste à calculer le coût minimal de production d'au moins V à partir de la période $i + 1$ sachant que la machine de production ne produit pas durant la période i .

Ici, on veut calculer, une valeur $CostMin_{i,Z,V}$, qui, pour tout $i = 0, \dots, N$, va nous donner la valeur du coût minimal d'une production d'au moins V , $V = 0, \dots, PMAX$, dès lors que l'on démarre à la période i , avec la micro-usine en état $Z = \{0, 1\}$. On ne tient pas compte de la capacité de la citerne. La période -1 est une période fictive qui correspond à l'état initial de la machine de production. $PMAX$ est fourni :

- Soit comme associé à une tournée pré-calculée, $PMAX$ est alors la quantité $\sum_{q=1}^Q u_q - E_0$. Q est le nombre de recharge et u_q la quantité chargée pendant la q^{ieme} recharge ;
- Soit comme estimée indépendamment de cette tournée, $PMAX$ est alors $2 \times \sum_{j=0}^M e_j$.

Entrées / Sorties de « Cout minimal production »

Les entrées sont :

- $N, Cost^F, R_i, Cost_i^V$;
- $PMAX \sim$ la quantité maximale à produire avant la période N ;
- $\bar{q}_i \sim$ la quantité maximum d'hydrogène qu'on peut produire entre les périodes i et $N - 1$ incluses.
 $\bar{q}_i = \sum_{k \geq i} R_k$.

La sortie est :

- $CostMin_{i,Z,V} \sim$ la valeur du coût minimal d'une production V , $V = 0, \dots, PMAX$, dès lors que l'on démarre la production à la période i , Z est l'état de la micro-usine à la fin de la période $i - 1$.
- $Z_{-1} = 0$ qui signifie que l'usine est à l'arrêt sur la période fictive -1 .

Algorithme « Cout minimal production »

Pour calculer les valeurs du vecteur $CostMin_{i,Z,V}$ on procède en backward, on commence à la période $N - 1$. Dans l'algorithme (5), nous commençons par remplir les cases $i = N$ car on connaît qu'à la fin du processus soit on a atteint la production demandée et on ne produit plus, soit on ne peut pas produire en N périodes la quantité demandée. $CostMin_{i,Z,V}$ réponds à la question comment produire au moins V unités d'hydrogène au coût le plus faible à partir de la période i .

On a présenté les mécanismes de filtrage qui permettent de diminuer le nombre d'états générés par le **DPS_SMEPC**. Le mécanisme de filtrage basée sur la borne supérieure/inférieure a besoin de la valeur d'une solution connue du problème **SMEPC**. On va calculer cette solution à l'aide d'une heuristique qui sera présentée à la section suivante. Cette heuristique est une adaptation gloutonne du **DPS_SMEPC**

Algorithme 5 Cout minimal production

Entrées: $N, PMAX, R, \bar{q}, Cost^F, Cost^V,$ **Sorties:** $CostMin$

Initialisation :

```

1: pour  $Z = 0$  à 1 faire
2:    $CostMin_{N,Z,0} \leftarrow 0$ ; /* Produire 0 coûte 0 */
3:   pour  $V = 1$  à  $PMAX$  faire
4:      $CostMin_{N,Z,V} \leftarrow +\infty$ ;
5:   fin pour
6: fin pour

```

Boucle principale :

```

7: pour  $i = (N - 1)$  à 0 faire
8:   pour  $Z = 0$  à 1 faire
9:     pour  $V = 1$  à  $PMAX$  faire
10:    Si ( $V > \bar{q}_i$ ) alors
11:       $CostMin_{i,Z,V} \leftarrow +\infty$ ; /* Impossible de produire V en commençant à la période i */
12:    sinon
13:       $P1 \leftarrow Sup(V - R_i, 0)$ ; /* La micro-usine produis */
14:       $C1 \leftarrow Cost_i^v + (1 - Z) \times Cost^F$ ;
15:       $CostMin_{i,Z,V} \leftarrow Inf(C1 + CostMin_{i+1,1,P1}, CostMin_{i+1,0,V})$ ; /* CostMin_{i+1,0,V} correspond à je ne produis pas */ /* CostMin_{i,Z,V} est le coût minimum de production si on produit à partir de la période i */
16:    Fin si
17:   fin pour
18:   fin pour
19: fin pour

```

Borne supérieure calculée par une adaptation gloutonne du DPS_SMEPC

On désignera par Greedy-SMEPC l'algorithme de calcule d'une borne supérieure par une heuristique qui est une adaptation gloutonne du **DPS_SMEPC**. La transformation du **DPS_SMEPC** en un algorithme glouton afin d'obtenir une borne supérieure W^{Curr} peut être réalisée en utilisant notre schéma de programmation dynamique et en effectuant un parcours en avant dans le réseau d'états du **SMEPC**, en choisissant à chaque pas de temps au sens de la programmation dynamique l'état qui à la meilleur valeur LB (décrise à la section précédente). On continuera le processus de prise de décision unique ment avec l'état sélectionné.

Néanmoins, nous devons veiller à éviter les stratégies qui verraient à la fois la micro-usine attendre de meilleurs prix $Cost_i^V$, de meilleures rendement de production R_i , et le véhicule attendre à la micro-usine pour une meilleure opportunité de chargement. Comme notre capacité à anticiper les incohérences liées à un manque de temps ou à un manque de capacité de recharge en hydrogène ne découle que de dispositifs d'approximation, il existe un risque que de telles stratégies d'attente fassent échouer notre processus de calcul d'une solution possible. Donc, afin de faire diminuer ce risque de défaillance, nous interdisons :

- Les décisions ($z = 0, x = 1$) liées à des situations où $T == i$ et où le réservoir de la micro-usine n'est pas suffisamment chargé pour permettre de faire le plein du véhicule ;
- Les décisions ($z = 0, x = 0$) liées à des situations où $T << i$ et $(p.i - T) \leq d_j$, et où le réservoir de la micro-usine n'est pas suffisamment chargé pour permettre de faire le plein du véhicule ;
- Décisions ($z = 0, x = 0, \delta = 0$) liées à des situations où $T << i$ et $(p.i - T) \geq d_j$, le véhicule décide de continuer sa tournée sans produire.

L'algorithme Greedy-SMEPC est décrit à l'algorithme (6). L'instruction 6 est exécutée telle que :

- La décision (z, x, δ) n'est pas interdite selon les mécanismes d'interdiction ci-dessus ;
- L'état résultant $s_1 = (Z_1, T_1, V_1^{Tank}, V_1^{Veh})$ n'est pas tué au temps résultant (i_1, j_1) par les mécanismes de filtrage logique ci-dessus ;
- $W + Transition_Cost + LB((i_1, j_1), s_1)$ est le plus petit possible, où *Transition_Cost* signifie le coût de la transition induite par l'application de (z, x, δ) à s au temps (i, j) au sens de la programmation dynamique.

Dans l'heuristique décrite par l'algorithme (6), on sélectionne à chaque pas de temps au sens de la programmation dynamique un unique état pour continuer le processus de prise de décisions. Lors de la phase expérimentation de cet algorithme, on a remarqué qu'on avec beaucoup de cas d'échec c'est-à-dire qu'on obtient aucune solution pour une grande partie des instances. On peut alors au lieu de sélectionner un seul état, sélectionner $NS > 1$ états ayant les meilleures valeurs LB pour continuer le processus de prise de décisions, cette procédure sera appelée **Greedy-SMEPC(NS)**. Ou alors, on peut sélectionner $NS > 1$ états ayant les meilleures valeurs LB et $BS > 1$ états ayant les pires valeurs LB pour continuer le processus de prise de décisions.

5.4.4 Mécanismes de filtrage heuristique

Notre objectif dans cette section est de décrire des mécanismes de filtrage heuristique qui serviront à diminuer le nombre d'états de l'algorithme général du **DPS_SMEPC** quitte à induire un certain niveau d'approximation. On rappelle qu'un état d'un temps (i, j) est défini par un quadruplet $(Z, T, V^{Tank}, V^{Veh})$, avec :

Algorithme 6 Greedy_SMEPC

Entrées: $N, M, TMax, H_0, E_0, C^{Tank}, C^{Veh}$,**Sorties:** Sol_Greedy, Val_Greedy **Initialisation :**

- 1: $i \leftarrow 0; j \leftarrow 0; W \leftarrow 0; Notstop;$
- 2: $S(0, 0) \leftarrow \{(s_0 = (0, 0, H_0, E_0), W = 0, Father = Indfini)\};$
- 3: $Sol_Greedy \leftarrow Undefined; Val_Greedy \leftarrow +\infty;$

Boucle principale :

- 4: **Tant que** $Notstop$ faire
 - 5: Définissez l'état actuel $s = (Z, T, V^{Tank}, V^{Veh})$, la valeur correspondante = W et la paire de temps actuelle = (i, j) ;
 - 6: Prenez une décision réalisable (au sens de la section 5.2.3) (z, x, δ) ;
 - 7: **Si** (z, x, δ) n'existe pas **alors**
 - 8: Stop; /* Echec : l'algorithme n'a pas trouvé de solution */
 - 9: $Sol_Greedy \leftarrow Undefined; Val_Greedy \leftarrow +\infty;$
 - 10: **sinon**
 - 11: **Si** $j_1 = M + 1$ **alors**
 - 12: Stop; /* Succès : l'algorithme a trouvé une solution */
 - 13: $Val_Greedy = W + Transition_Cost;$
 - 14: $Sol_Greedy = (s_1, (s, i, j));$
 - 15: **sinon**
 - 16: Exécuter la décision (z, x, δ) : mettre à jour i,j,s et la valeur W associée ;
 - 17: **Fin si**
 - 18: **Fin si**
 - 19: **Fin tant que**
-

- Z : l'état de la micro-usine de production d'hydrogène ;

$$Z = \begin{cases} 1 & \text{si la micro-usine est active à la fin de la période } i - 1. \\ 0 & \text{sinon.} \end{cases}$$

- V^{Tank} : la quantité d'hydrogène dans la citerne à la micro-usine au début de la période i ;
- V^{Veh} : la quantité d'hydrogène dans le réservoir du véhicule lorsqu'il arrive à la station j ;
- $T \in 0, \dots, TMax$ est une date signifiant :
 - Si $T >> i$ alors le véhicule est en route vers la station j et il y arrivera à la date T (Voir figure (5.1)) ;
 - Si $T << i$ alors le véhicule est entre la station j et la micro-usine. Le véhicule peut être entraîné à attendre à la micro-usine pour être rechargeé (Voir figure (5.1)) ;
 - Si $T == i$ alors le véhicule se trouve à la station j , et doit choisir entre aller à la station $j + 1$ ou aller à la micro-usine se recharger (Voir figure (5.1)).

On fixe un paramètre entier K dit de fluidification (on pourra utiliser par exemple $K = 10$ ou $K = 5$). Pour chaque station j , on pose :

- $\Delta_j = \sum_{0 \leq s \leq j-1} d_s$
- $\Delta_j^* = TMax - \sum_{j \leq s \leq M} d_s$
- $\Pi_j = \Delta_j^* - \Delta_j$.

Lorsqu'on est à une paire de temps (i_0, j_0) et que l'on vient de générer, à partir d'un état E_0 et d'une décision D un nouvel état $E = (Z, T, V^{Tank}, V^{Veh})$ accompagné d'une valeur W et associé au temps (i, j) alors :

1. On applique tous les mécanismes de filtrage décrit précédemment ;
2. Si l'état E n'a pas été supprimé par ces mécanismes de filtrage alors on balaie l'ensemble des états du temps (i, j) et chaque fois que l'on rencontre un état $E_1 = (Z_1, T_1, V_1^{Tank}, V_1^{Veh})$ accompagné d'une valeur W_1 , on procède comme suit :

Si $Z = Z_1$ et $|V^{Tank} - V_1^{Tank}| \leq (C^{Tank}/K)$ et $|V^{Veh} - V_1^{Veh}| \leq (C^{Veh}/K)$ et $|T - T_1| \leq (\Pi_j/K)$ alors on dit que E et E_1 sont équivalents modulo K :

- si $W < W_1$ alors on remplace E_1, W_1 par E, W dans les états du temps (i, j) ;
- dans le cas inverse, on ne fait rien, donc on n'insère pas E, W parmi les états du temps (i, j) ;

On vient de présenter l'algorithme général du **DPS_SMEPC**, les mécanismes de filtrage de cet algorithme et un algorithme qui calcule une borne supérieure pour le **DPS_SMEPC**. Dans la section suivante, on va mesurer l'impact des mécanismes de filtrage sur le nombre d'états générés par le schéma de programmation dynamique **DPS_SMEPC** qui résout le problème **SMEPC** et l'efficacité de l'algorithme Greedy-DPS par rapport à l'algorithme **DPS_SMEPC**.

5.5 Description d'une heuristique rapide *He* pour résoudre SMEPC

Dans cette section, on va décrire une heuristique rapide nommée *He* qui calcule une solution d'une instance du problème SMEPC très rapidement. Nous proposons ici une nouvelle modélisation

du problème du véhicule dans laquelle la micro-usine nommée noeud 0 est distincte du dépôt nommé noeud D . Nous avons 3 types d'arc mais leur signification est totalement définie par leur origine. Un arc dont l'origine est j , $j = 0, \dots, M$ signifie que "le véhicule fait le plein de carburant après la station j " (le véhicule fait le plein après le dépôt dans le cas $j = 0$). Dans les sous-sections suivantes, nous expliquons comment le problème VD peut être modélisé par un graphe orienté.

5.5.1 Le graphe

Le problème VD peut être modélisé en un problème de plus court chemin dans un graphe G . Si on suppose que le véhicule fait le plein à chaque recharge, VD peut être modélisé sous forme d'un graphe orienté $G = (V, E)$ qui se construit de la façon suivante :

Son ensemble V de sommets est l'ensemble $\{D, 0, 1, \dots, M, M + 1\}$. D et $M + 1$ représentent respectivement le dépôt au début et à la fin du trajet du véhicule. Un noeud j , $j = 0, \dots, M$ correspond au fait que le véhicule se trouve à la micro-usine après avoir quitté la station j , ou juste après avoir quitté le dépôt si $j = 0$.

Dans E , il existe trois types d'arcs :

- **Arcs de type 1** : Un arc $(j, j') \in E$, $0 \leq j < j' \leq M$, indique que le véhicule est capable de se rendre de la micro-usine (après la station j) à la micro-usine (après la station j') sans opération de recharge. Cet arc existe si l'inéquation (5.1) est vérifiée. Cette inéquation signifie que la quantité d'énergie dont a besoin le véhicule pour réaliser ce trajet, ne doit pas être supérieure à la capacité maximale du véhicule.

$$\varepsilon_{j+1}^* + \sum_{j+1 \leq k \leq j'-1} e_k + \varepsilon_{j'} \leq C^{Veh} \quad \forall j < j' = 0, \dots, M \quad (5.1)$$

- **Arcs de type 2** : Un arc $(D, j) \in E$, $0 \leq j \leq M$, signifie que :

- le véhicule peut aller du dépôt à la micro-usine juste après le dépôt (si $j = 0$) ;
- le véhicule peut aller du dépôt à la station j sans faire le plein (si $j \geq 1$).

L'énergie initiale E_0 doit être suffisante pour effectuer ce trajet. Par conséquent, cet arc existe si l'inéquation (5.2) est vérifiée.

$$\sum_{k=0, \dots, j-1} e_k + \varepsilon_j \leq E_0 \quad \forall j = 0, \dots, M \quad (5.2)$$

- **Arcs de type 3** : Les arcs $(j, M + 1) \in E$, $0 \leq j \leq M$ signifient que le véhicule peut aller de la micro-usine après la station j , jusqu'au dépôt final sans faire le plein. De plus, le véhicule doit arriver au dépôt avec au moins son énergie initiale E_0 . Cet arc existe si l'inéquation (5.3) est vérifiée.

$$\varepsilon_{j+1}^* + \sum_{k=j+1, \dots, M} e_k + E_0 \leq C^{Veh} \quad \forall j = 0, \dots, M \quad (5.3)$$

5.5.2 Les poids sur les arcs

Soit $P = (D, j_1, \dots, j_q, M + 1)$ un chemin entre D et $M + 1$ dans G . La longueur de P est égale à :

$$\sum_{k=0, \dots, j_1-1} t_k + d_{j_1} + d_{j_1+1}^* + p + \sum_{k=j_1+1, \dots, j_2-1} t_k + d_{j_2} + d_{j_2+1}^* + p + \dots + \sum_{k=j_{q-1}+1, \dots, j_q-1} t_k + d_{j_q} + d_{j_q+1}^* + p + \sum_{k=j_q+1, \dots, M} t_k = \sum_{k=0, \dots, M} t_k + d_{j_1} + d_{j_1+1}^* + p - t_{j_1} + d_{j_2} + d_{j_2+1}^* + p - t_{j_2} + \dots + d_{j_q} + d_{j_q+1}^* + p - t_{j_q}$$

Nous désignons par $detour(j)$ la distance supplémentaire (ou le temps supplémentaire) parcourue par le véhicule pour se recharger après la station j : $\forall j \in \{0, \dots, M\}$, $detour(j) = d_j + d_{j+1}^* + p - t_j$

Ainsi, attribuons le poids ou le coût suivant à chaque arc (j, j') de G .

arcs de type 1 et arcs de type 3 : leur coût est $detour(j)$.

arcs de type 2 : leur coût est arbitrairement pris égal à 0.

Les arcs de type 1 et 3 peuvent être fusionnés en fixant $\varepsilon_{M+1} = E_0$. En bref, nous avons pour les arcs de type 1 et 3 (j, j') :

$\forall j, j', 0 \leq j < j' \leq M+1$ et $\varepsilon_{j+1}^* + \sum_{k=j+1, \dots, j'-1} e_k + \varepsilon_{j'} \leq C^{Veh}$. Et le poids de (j, j') est $detour(j)$.

Le problème du véhicule consiste à déterminer le plus court chemin dans le graphe G .

5.5.3 Exemple

Considérons l'instance suivante de la figure (5.5). Les coûts sur les arcs sont (temps, énergie). La charge initiale d'hydrogène dans le réservoir du véhicule E_0 est de 20 unités et sa capacité C^{Veh} est de 30 unités. Le graphe G pour la modélisation du problème du véhicule est illustré à la figure (5.6).

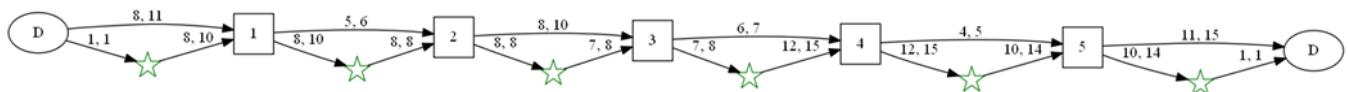


FIGURE 5.5 – Une instance

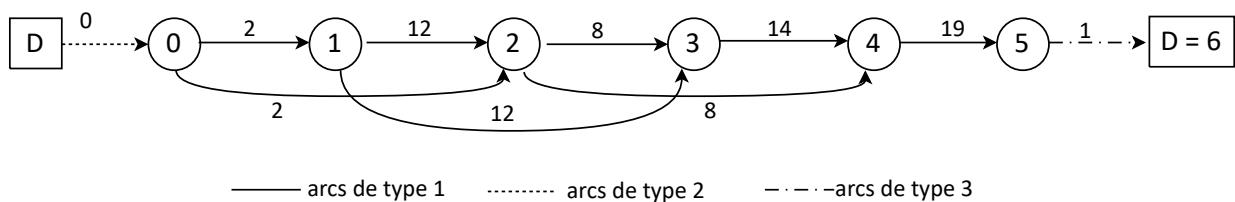


FIGURE 5.6 – Le graphe G correspondant à l'instance de la figure (5.5) .

Le plus court chemin est : $D \rightarrow 0 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$ de coût 30 et cela signifie que le véhicule fait le plein après le dépôt initial et après les stations 2, 4 et 5. Pour chaque recharge, on peut calculer la quantité minimale d'hydrogène que le véhicule doit recharger pour que qu'il fasse le plein afin de pouvoir aller à la prochaine recharge (ou dépôt final). Il en résulte que :

- A la première recharge, le véhicule arrive à la micro-usine avec 19 (qui correspond à $E_0 - \varepsilon_0$) unités d'hydrogène dans son réservoir et il lui en faut 24 pour aller à la prochaine recharge. Il doit donc recharger 5 unités pour faire le plein.
- A la deuxième recharge, le véhicule arrive à la micro-usine avec 0 unité d'hydrogène dans son réservoir et il a besoin de 30 pour aller à la prochaine recharge. Il doit donc recharger 30 unités pour faire le plein.
- A la troisième recharge, le véhicule arrive à la micro-usine avec 0 unité d'hydrogène dans son réservoir et il lui en faut 28 pour aller à la prochaine recharge. Il doit donc recharger 28 unités pour faire le plein.

- A la quatrième recharge, le véhicule arrive à la micro-usine avec 0 unité d'hydrogène dans son réservoir et il lui en faut 21 pour arriver au dépôt final avec $E_0 = 20$ unités d'hydrogène dans son réservoir. Il doit donc recharger 21 unités pour faire le plein.

Ainsi, nous pouvons calculer la quantité totale H d'hydrogène nécessaire pour toutes les recharges du véhicule.

Pour pouvoir comparer l'heuristique précédente aux autres algorithmes qui résolvent SMEPC, il faudrait calculer un calendrier de production, c'est-à-dire décider pendant quelles périodes la micro-usine produit. Pour ce faire, nous connaissons déjà la quantité totale H d'hydrogène nécessaire à toutes les recharges du véhicule. Maintenant, nous faisons produire la micro-usine jusqu'à ce qu'au moins H unités d'hydrogène aient été générées. Plusieurs contraintes doivent être respectées :

- La micro-usine ne peut pas produire lorsque le véhicule fait le plein ;
- La capacité du réservoir de la micro-usine ne doit jamais être dépassée, ;
- Le véhicule ne peut faire le plein que si la quantité d'hydrogène dans le réservoir de la micro-usine est suffisante : le véhicule attend que cette quantité soit atteinte.

Partant de la solution du problème du véhicule, pour calculer le planning de production, on utilise deux méthodes :

- La première consiste à produire le plus tôt possible l'énergie H que le véhicule viendra recharger ;
- La seconde consiste à trouver la meilleure façon de produire entre deux recharges, c'est-à-dire au coût le plus bas pour satisfaire la recharge.

Si la quantité d'énergie produite est insuffisante pour satisfaire le recharge, nous essayons de faire attendre le véhicule et de produire uniquement ce dont nous avons besoin pour la prochaine recharge. On compare les solutions obtenues par ces deux méthodes et on considère que la meilleure solution est celle de l'heuristique rapide qu'on nomme He .

5.6 Expérimentations numériques

5.6.1 Objectifs et contexte technique

Notre objectif est d'obtenir une évaluation de l'impact des dispositifs de filtrage décrits et de la qualité des procédures gloutonnes décrites. Cet impact est à double sens : Premièrement, Il fait diminuer le nombre d'états qui sont traités tout au long du processus. Deuxièmement, l'utilisation du filtrage heuristique peut induire un gap par rapport à l'optimalité.

Les algorithmes ont été implémentés en C++ et l'IDE utilisé est Visual Studio 2017. Les expérimentations sont réalisées sur un ordinateur équipé d'un processeur AMD EPYC 7H12 64-Core, 512 Go de RAM et fonctionnent sous Gnu/linux Ubuntu 20.04.2. Le temps maximum du CPU est fixé à 1 heure.

5.6.2 Instances

Ici, nous utilisons un ensemble de 50 instances, dont les caractéristiques sont décrites par les tableaux (4.4) et (4.3).

Soient les instances A et B illustrées respectivement aux figures (5.7), (5.8) et (5.10), (5.11), on a illustré les solutions obtenues par l'algorithme DPS_SMEPC. Les solutions des instances A et B sont respectivement illustrées par les figures (5.9) et (5.12).

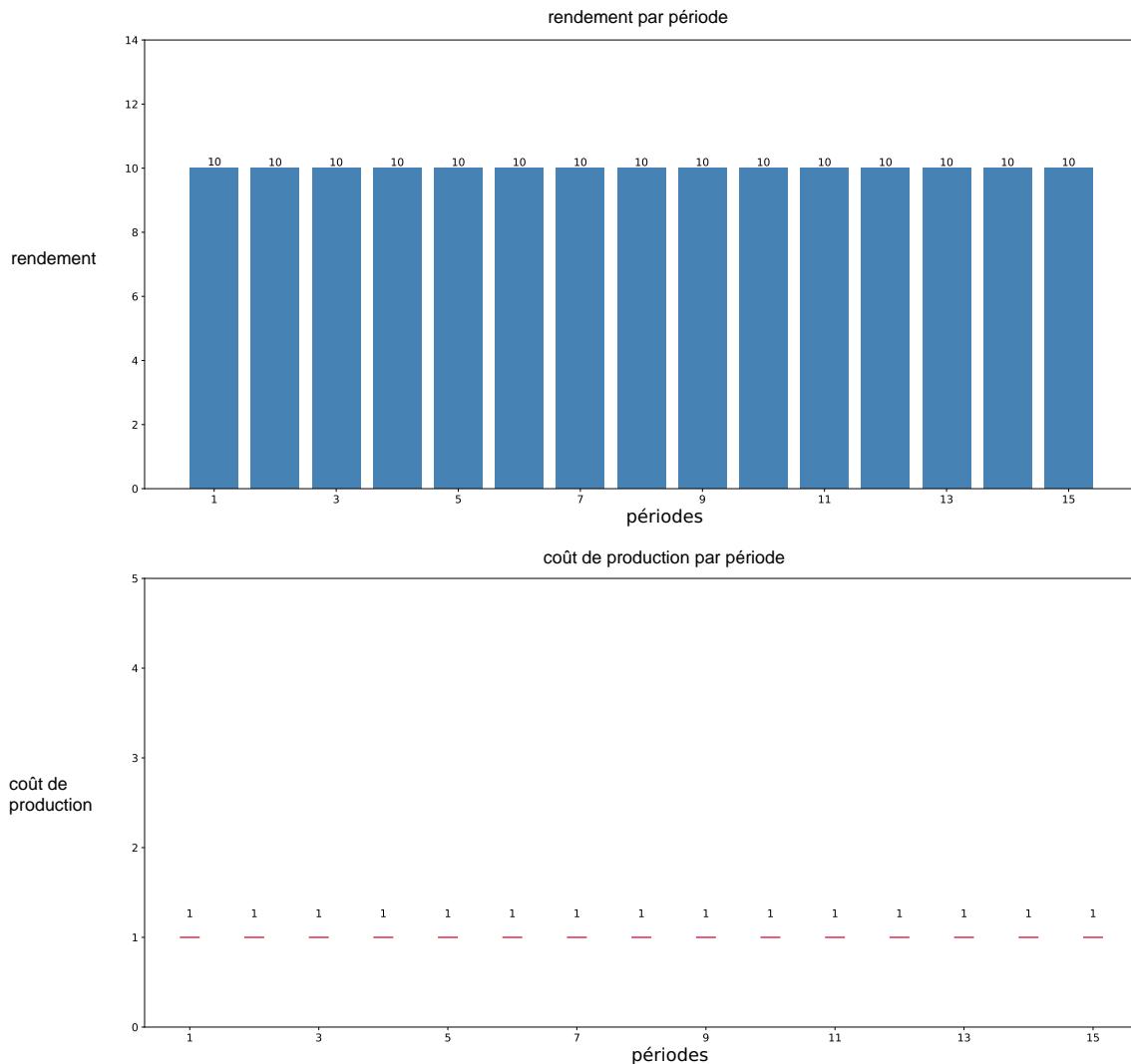


FIGURE 5.7 – Les données de la partie production de l’instance A. Sur chaque période, le rendement est 10 et le coût de production est 1.

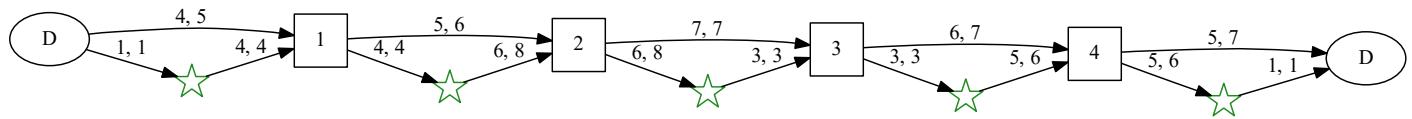


FIGURE 5.8 – Les données de la partie recharge de l’instance A. Les valeurs sur les arcs sont de la forme (durée, énergie).

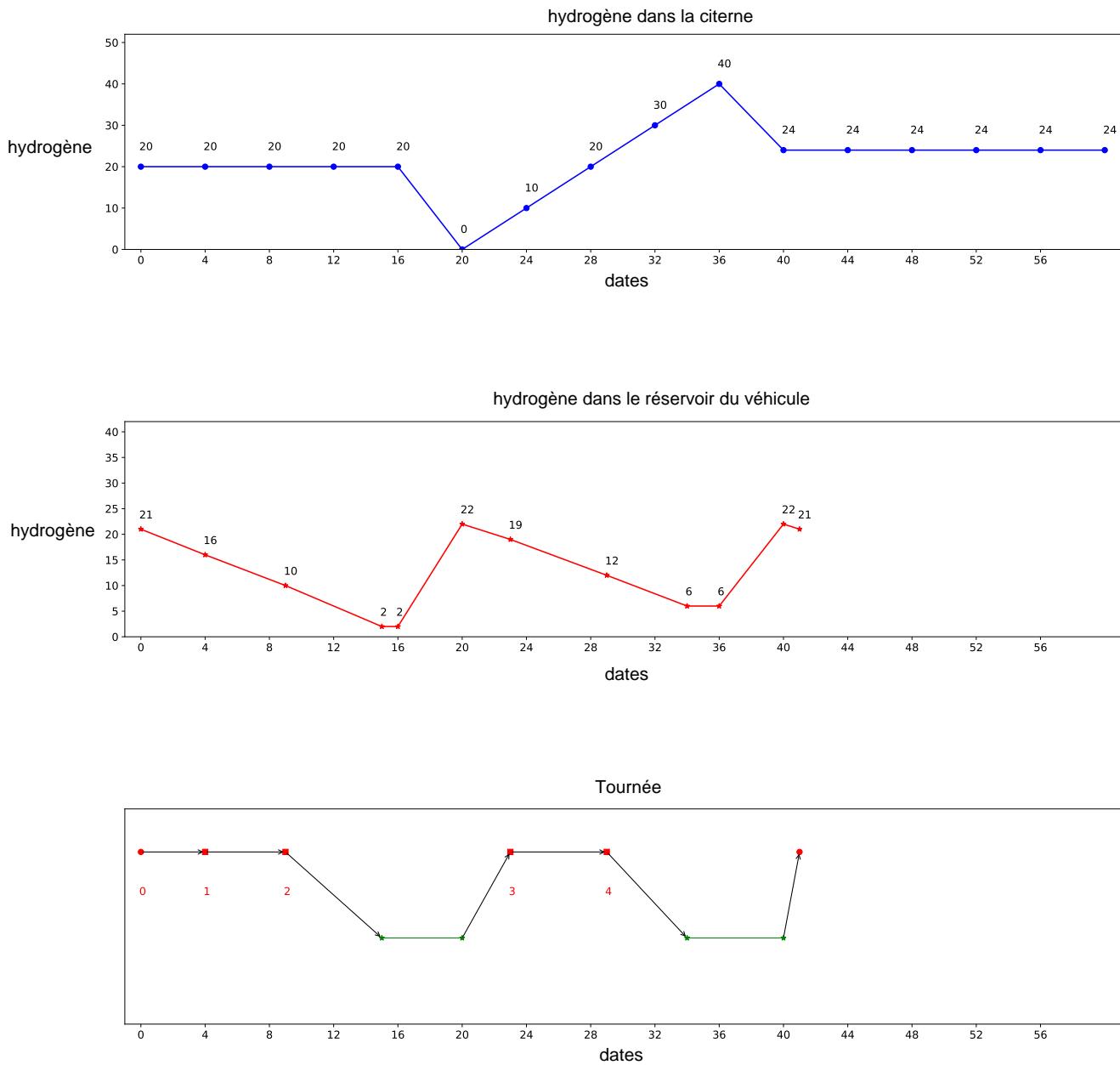


FIGURE 5.9 – Solution de l’instance A : on a deux recharges l’une entre la station 2 et la station 3 et l’autre entre la station 4 et le dépôt final.

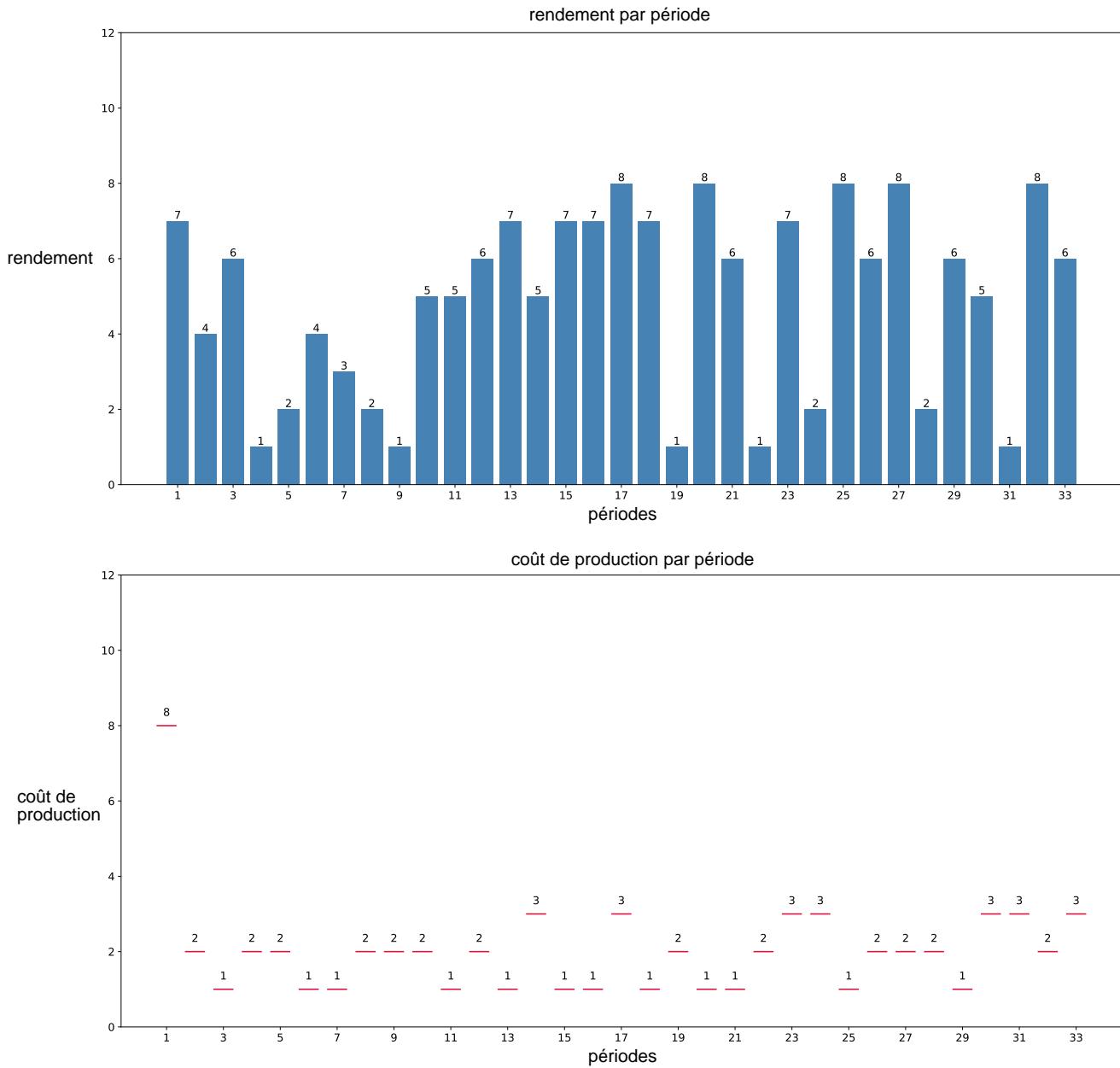


FIGURE 5.10 – Les données de la partie production de l’instance B.

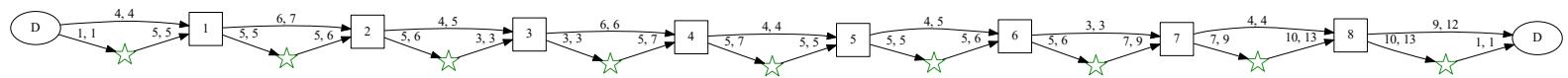


FIGURE 5.11 – Les données de la partie recharge de l’instance B. Les valeurs sur les arcs sont de la forme (durée, énergie).

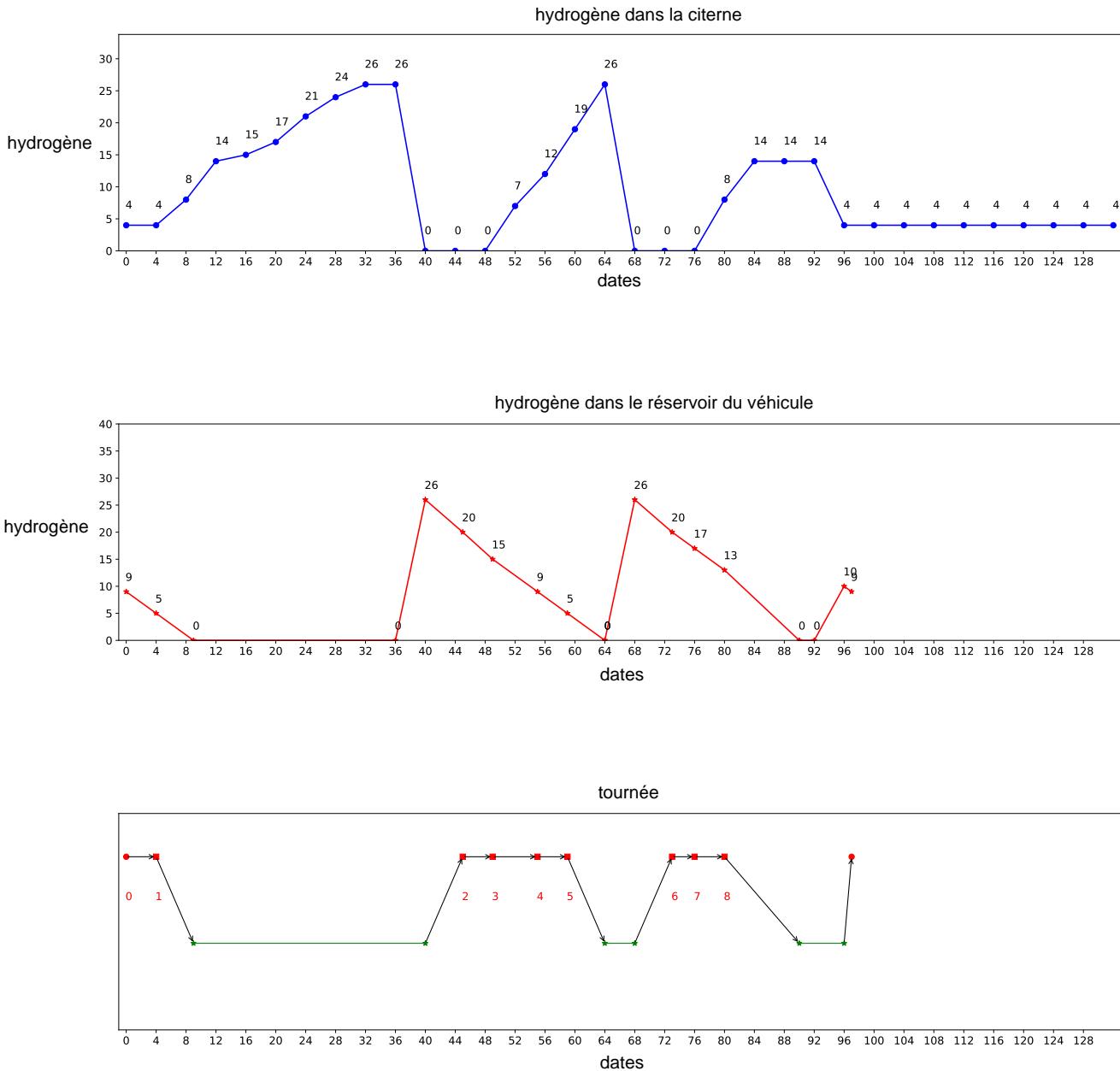


FIGURE 5.12 – Solution de l’instance B : on a trois recharges.

5.6.3 Recherche d’une borne supérieure de qualité pour le DPS_SMEPC

L’algorithme (7) décrit la méthode de recherche de la borne supérieure pour le DPS_SMEPC.

Pour pouvoir exécuter l’algorithme DPS_SMEPC avec le filtrage par estimation optimiste on a besoin de connaître une borne supérieure. Pour calculer cette borne supérieure, on va exécuter plusieurs algorithmes et sélectionner la valeur la plus petite comme borne supérieure. Le premier algorithme qu’on exécute est l’heuristique rapide de la section 5.5. Ensuite, on exécute l’algorithme

Algorithme 7 Search_BSUP

Sorties: BSUP

- 1: Exécuter l'heuristique rapide **He** et mettre sa valeur dans $BSUP_1$;
 - 2: Exécuter l'heuristique **Greedy-SMEPC(20)** avec comme borne supérieure $BSUP_1$ et mettre sa valeur dans $BSUP_2$;
 - 3: Exécuter l'heuristique **Greedy-SMEPC(50)** avec comme borne supérieure $Inf(BSUP_1, BSUP_2)$ et mettre sa valeur dans $BSUP_3$;
 - 4: Exécuter l'heuristique **Greedy-SMEPC(100)** avec comme borne supérieure $Inf(BSUP_1, BSUP_2, BSUP_3)$ et mettre sa valeur dans $BSUP_4$;
 - 5: $BSUP \leftarrow Inf(BSUP_1, BSUP_2, BSUP_3, BSUP_4,)$;
-

Greedy-SMEPC(NS) avec $NS = 20$, en utilisant comme borne supérieure la valeur de l'heuristique rapide. Puis, on exécute **Greedy-SMEPC(NS)** avec $NS = 50$, en utilisant comme borne supérieure la meilleure valeur entre $NS = 20$ et l'heuristique rapide. Enfin, on exécute **Greedy-SMEPC(NS)** avec $NS = 100$, en utilisant comme borne supérieur la meilleure valeur entre $NS = 20$, $NS = 50$ et l'heuristique rapide. Au final, la borne supérieure qui est utilisée pour le schéma de programmation dynamique est la meilleure valeur parmi les valeurs des algorithmes suivants : **Greedy-SMEPC(20)**, **Greedy-SMEPC(50)**, **Greedy-SMEPC(100)** et l'heuristique rapide.

Nous voulons obtenir une évaluation de la qualité de la procédure gloutonne **Greedy-SMEPC(NS)** décrite plus haut et de l'heuristique rapide présenté plus haut. Les tableaux (5.3) et (5.4) présentent respectivement les valeurs, les gaps et les temps CPU des instances du paquet INST_CTE et du paquet INST_VAR. La signification des colonnes est :

- Obj** est la valeur obtenue ;
- Gap** est le gap par rapport à l'optimalité $Gap = 100 \times \frac{Obj - opt}{opt}$; où Obj est la valeur obtenue et opt est la valeur optimale obtenue à l'aide du modèle $MILP_{SMEPC}$ du chapitre précédent. Si une instance n'a pas été résolu jusqu'à l'optimalité, on utilise la borne supérieure de l'exécution sur 8 threads avec un temps limite de 3 heures (voir tableau (4.7) et (4.8)).
- CPU** est le temps d'exécution en secondes.

On recherche ici la meilleure borne supérieure qu'on utilisera lors de l'exécution de l'algorithme DPS_SMEPC pour diminuer le nombre d'états. Pour cela, on exécute plusieurs algorithmes. Et la borne supérieure sera la meilleure valeur obtenue. La première procédure qu'on exécute est l'heuristique rapide **He**. En analysant les résultats des instances du paquet INST_CTE, on remarque que l'algorithme **He** donne toujours une solution. En moyenne, parmi les valeurs de NS testées, la procédure la plus rapide est **Greedy-SMEPC(20)** et la procédure la plus lente est **Greedy-SMEPC(100)**, ce qui est normal car on garde beaucoup plus d'états.

Pour le paquet d'instances INST_CTE et le paquet d'instances INST_VAR on fait les remarques suivantes :

- On remarque que lorsqu'on augmente la valeur de NS , on obtient de meilleures solutions, ce qui est sûrement dû au fait que la valeur de la procédure exécutée précédemment est utilisée comme borne supérieure de la procédure suivante.
- On constate qu'il y a une amélioration de la valeur et cela s'accompagne d'une augmentation du temps de calcul, car on augmente le nombre d'états qu'on garde.
- La rapidité d'exécution de la procédure **Greedy-SMEPC(NS)** peut être dû à l'efficacité de l'heuristique rapide **He**.

Recherche d'une borne supérieure de qualité pour le DPS_SMEPC : les instances du paquet INST_VAR

Le tableau (5.3) présente les résultats du calcul d'une borne supérieure de qualité pour les instances du paquet INST_VAR. Le temps d'exécution de tous les algorithmes est toujours inférieur à 0,7 seconde.

num	He			NS(20)			NS(50)			NS(100)		
	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU
1	169	29,01	0,01	141	7,63	0,00	131	0,00	0,00	131	0,00	0,00
2	199	31,79	0,01	171	13,25	0,00	155	2,65	0,00	155	2,65	0,00
3	173	20,14	0,01	152	5,56	0,00	149	3,47	0,00	144	0,00	0,00
4	211	50,71	0,01	198	41,43	0,00	160	14,29	0,00	155	10,71	0,00
5	194	20,50	0,01	184	14,29	0,00	172	6,83	0,00	161	0,00	0,00
6	237	33,15	0,01	220	23,60	0,00	209	17,42	0,00	199	11,80	0,01
7	258	16,22	0,00	226	1,80	0,00	222	0,00	0,00	222	0,00	0,00
8	240	25,00	0,00	219	14,06	0,00	207	7,81	0,00	200	4,17	0,01
9	715	11,02	0,00	702	9,01	0,00	683	6,06	0,01	680	5,59	0,01
10	1231	8,08	0,00	1212	6,41	0,00	1191	4,57	0,01	1178	3,42	0,01
11	181	35,07	0,00	168	25,37	0,00	154	14,93	0,00	140	4,48	0,01
12	1033	13,27	0,00	1014	11,18	0,01	990	8,55	0,02	970	6,36	0,03
13	1017	6,38	0,00	1006	5,23	0,01	992	3,77	0,01	970	1,46	0,02
14	1535	11,88	0,00	1535	11,88	0,01	1524	11,08	0,03	1513	10,28	0,06
15	1407	4,53	0,00	1402	4,16	0,01	1381	2,60	0,01	1371	1,86	0,02
16	1398	8,29	0,00	1390	7,67	0,01	1376	6,58	0,02	1361	5,42	0,04
17	1356	5,53	0,00	1343	4,51	0,03	1324	3,04	0,06	1318	2,57	0,11
18	2553	7,59	0,03	2551	7,50	0,03	2518	6,11	0,05	2490	4,93	0,09
19	2457	11,03	0,05	2449	10,66	0,04	2449	10,66	0,09	2449	10,66	0,16
20	2807	5,57	0,00	2780	4,55	0,03	2751	3,46	0,06	2733	2,78	0,12
21	1562	10,78	0,00	1557	10,43	0,06	1542	9,36	0,12	1529	8,44	0,22
22	1487	10,80	0,00	1482	10,43	0,06	1477	10,06	0,13	1465	9,17	0,26
23	2897	11,85	0,00	2885	11,39	0,06	2860	10,42	0,12	2837	9,54	0,23
24	3110	10,87	0,00	3094	10,30	0,07	3076	9,66	0,12	3058	9,02	0,22
25	2153	4,87	0,00	2151	4,77	0,08	2150	4,72	0,17	2147	4,58	0,34
26	2482	5,35	0,00	2475	5,05	0,03	2460	4,41	0,06	2459	4,37	0,10
27	2583	4,66	0,00	2564	3,89	0,12	2552	3,40	0,19	2543	3,04	0,36
28	3246	6,29	0,00	3243	6,19	0,11	3229	5,73	0,25	3213	5,21	0,50
29	4750	7,05	0,00	4750	7,05	0,07	4749	7,03	0,14	4740	6,83	0,28
30	3654	3,75	0,00	3642	3,41	0,13	3632	3,12	0,31	3621	2,81	0,60

TABLE 5.3 – Les instances du paquet INST_VAR : recherche d'une borne supérieure de qualité pour le DPS_SMEPC.

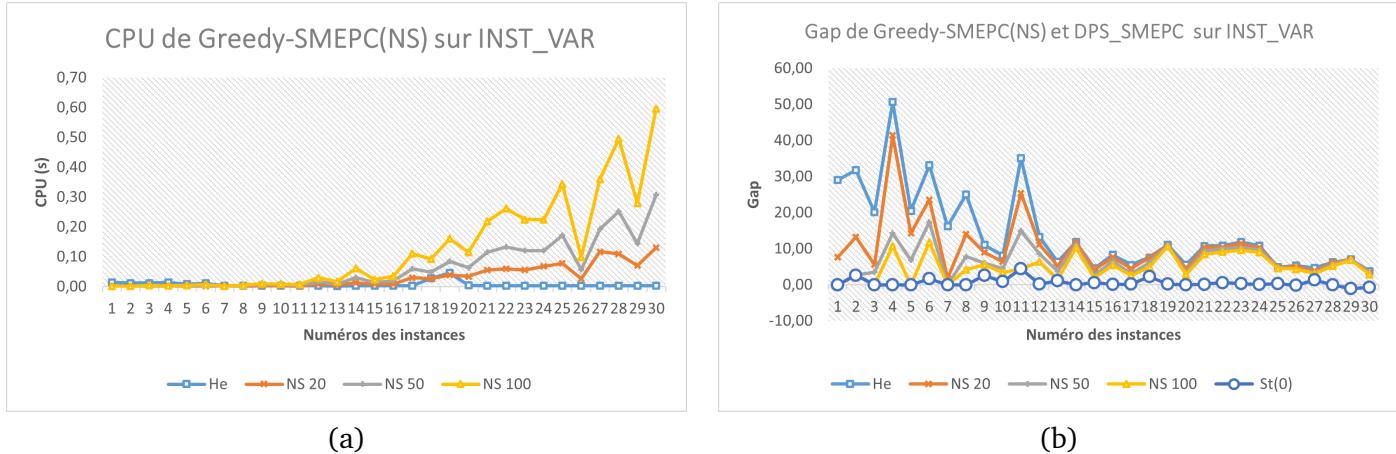


FIGURE 5.13 – Représentation graphique du tableau (5.3) et de $St(0)$. (a) représente le temps CPU et (b) représente le gap à l'optimalité de chaque instance de INST_VAR.

Recherche d'une borne supérieure de qualité pour le DPS_SMEPC : les instances du paquet INST_CTE

Consécutivement, le tableau (5.4) présente les résultats des instances du paquet INST_CTE. On a plus de la moitié des instances qui ont un gap inférieur à 25%. Le temps d'exécution de tous les algorithmes est toujours inférieur à 0,03 seconde.

num	He			NS(20)			NS(50)			NS(100)		
	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU
31	325	34,85	0,02	325	34,85	0,00	325	34,85	0,00	325	34,85	0,00
32	520	45,25	0,01	520	45,25	0,00	415	15,92	0,00	373	4,19	0,00
33	271	21,52	0,01	263	17,94	0,00	231	3,59	0,00	223	0,00	0,00
34	259	28,22	0,01	235	16,34	0,00	235	16,34	0,00	235	16,34	0,00
35	395	53,70	0,01	323	25,68	0,00	306	19,07	0,00	298	15,95	0,00
36	380	68,14	0,00	335	48,23	0,00	276	22,12	0,00	237	4,87	0,00
37	222	25,42	0,01	186	5,08	0,00	177	0,00	0,00	177	0,00	0,00
38	357	26,60	0,01	303	7,45	0,00	283	0,35	0,00	283	0,35	0,00
39	347	10,86	0,01	347	10,86	0,00	341	8,95	0,00	339	8,31	0,00
40	361	7,44	0,01	357	6,25	0,00	336	0,00	0,00	336	0,00	0,00
41	974	8,10	0,00	945	4,88	0,00	924	2,55	0,01	924	2,55	0,01
42	1619	12,43	0,00	1619	12,43	0,00	1593	10,63	0,01	1440	0,00	0,01
43	1623	5,66	0,00	1623	5,66	0,00	1623	5,66	0,01	1623	5,66	0,02
44	992	7,94	0,00	977	6,31	0,00	960	4,46	0,01	919	0,00	0,01
45	957	3,01	0,00	947	1,94	0,00	947	1,94	0,01	947	1,94	0,01
46	969	13,87	0,00	945	11,05	0,00	868	2,00	0,01	868	2,00	0,02
47	1170	24,20	0,00	1134	20,38	0,01	1016	7,86	0,01	942	0,00	0,02
48	1325	6,00	0,00	1325	6,00	0,01	1325	6,00	0,01	1325	6,00	0,02
49	886	6,62	0,00	878	5,66	0,01	856	3,01	0,01	831	0,00	0,02
50	1132	4,81	0,00	1132	4,81	0,01	1132	4,81	0,01	1084	0,37	0,02

TABLE 5.4 – Les instances du paquet INST_CTE : recherche d'une borne supérieure de qualité pour le DPS_SMEPC.

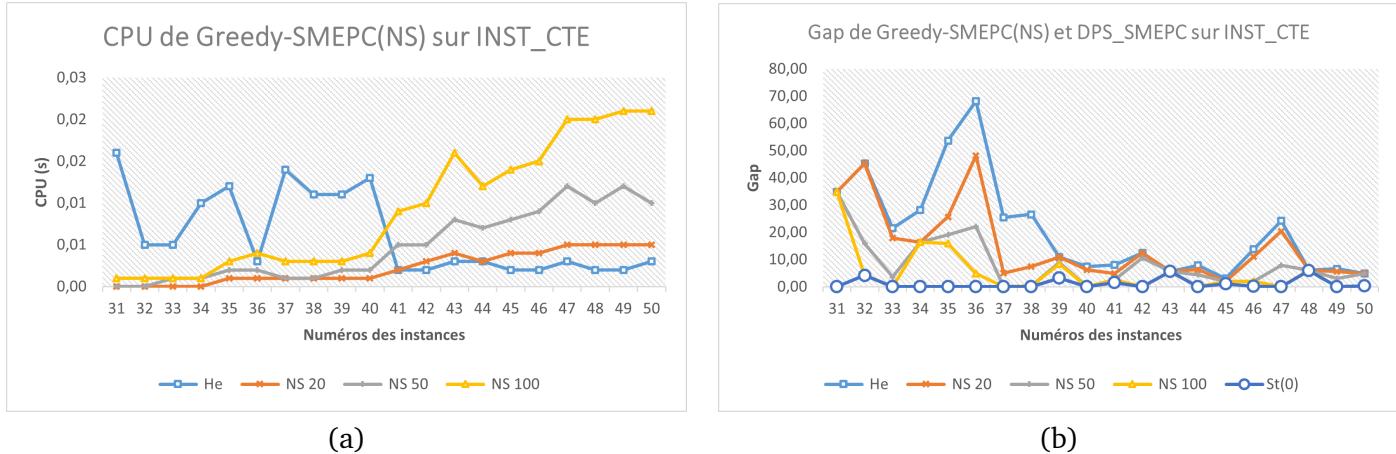


FIGURE 5.14 – Représentation graphique du tableau (5.4) et de $St(0)$. (a) représente le temps CPU et (b) représente le gap à l'optimalité de chaque instance de INST_CTE.

On vient de présenter le mécanisme de recherche d'une borne supérieure de qualité pour le DPS_SMEPC. La section suivante sera consacrée à la mesure de l'impact des mécanismes de filtrage du DPS_SMEPC sur le nombre d'états générés.

5.6.4 Impact des mécanismes de filtrage du DPS_SMEPC

Nous voulons obtenir une évaluation de la qualité du schéma de programmation dynamique DPS_SMEPC. On commence cette partie en rappelant tous les mécanismes de filtrage. Si on donne maintenant une paire de temps (i, j) et un état connexe $s = (Z, T, V^{Tank}, V^{Veh})$ avec la valeur W . Soient $Temps_{j,V^{Veh}}$ et $Energie_{j,V^{Veh}}$ qui représentent respectivement la quantité minimale de temps et d'énergie dont le véhicule a besoin pour se déplacer de la station j au dépôt final avec la quantité d'hydrogène V^{Veh} dans son réservoir. Afin de limiter le nombre d'états, on applique différents mécanismes de filtrage. En particulier, si on veut ajouter l'état s au temps (i, j) , celui-ci doit respecter certains mécanismes de filtrage. Les mécanismes de filtrages qui ont été implémentés sont les suivants :

1. Les mécanismes de filtrage logique :

- La durée de la tournée ne peut pas être supérieure au temps maximal : $[(i \times p \leq T) \wedge (T + Temps_{j,V^{Veh}} \leq TMax)] \vee [(i \times p > T) \wedge (i \times p + d_{j+1}^* + \sum_{j+1 \leq k \leq M} t_k \leq TMax)]$
- La quantité d'énergie utilisée par le véhicule lors de la tournée ne peut pas être supérieure à la somme de l'énergie stockée et de l'énergie qu'on peut produire : $[(i \times p \leq T) \wedge (V^{Tank} + \sum_{i \leq k \leq N-1} R_k + V^{Veh} \geq Energie_{j,V^{Veh}})] \vee [(i \times p > T) \wedge (V^{Tank} + \sum_{i \leq k \leq N-1} R_k + V^{Veh} \geq \varepsilon_{j+1}^* + \sum_{j+1 \leq k \leq M} e_k)]$

2. Le mécanisme de filtrage par estimation optimiste stipule qu'on supprime un état si ce dernier aboutira à une solution « moins bonne » qu'une solution calculée avec une heuristique rapide :

$$W + CostMin(i, Z, (Energie_{j,V^{Veh}} - V^{Tank} - V^{Veh})^+) + \beta \times Temps_{j,V^{Veh}} \leq BSUP,$$

où $BSUP$ est une borne supérieure calculée avec l'algorithme (7). La section précédente était consacrée à la présentation du mécanisme de recherche de la valeur $BSUP$.

3. Les mécanismes de filtrage heuristique stipulent que si on trouve dans (i, j) un état $s_1 = (Z_1, T_1, V_1^{Tank}, V_1^{Veh})$ (accompagné d'une valeur W_1) qui est équivalent modulo K à s alors si $W < W_1$ alors on remplace E_1, W_1 par E, W dans les états du temps (i, j) . Les états s et s_1 sont

équivalents modulo K si $Z = Z_1$ et $|V^{Tank} - V_1^{Tank}| \leq (C^{Tank}/K)$ et $|V^{Veh} - V_1^{Veh}| \leq (C^{Veh}/K)$ et $|T - T_1| \leq (\Pi_j/K)$. sachant que le paramètre K doit être préalablement prefixé.

Nous voulons ici obtenir une évaluation de l'impact des mécanismes de filtrage décrit ci-dessus. Cet impact est à double sens : premièrement, il fait diminuer le nombre d'états qui sont traités tout au long du processus; deuxièmement, le filtrage heuristique peut induire un gap entre la solution obtenue et la solution optimale. On exécute **DPS_SMEPC** de quatre manières :

1. Sans mécanismes de filtrage, on nomme ces résultats $St(1)$;
2. Avec les mécanismes de filtrage logique, on nomme ces résultats $St(2)$;
3. Avec les mécanismes de filtrage logique et les mécanismes de filtrage par estimation optimiste, on nomme ces résultats $St(3)$;
4. Avec les mécanismes de filtrage logique, les mécanismes de filtrage par estimation optimiste et les mécanismes de filtrage heuristique, on nomme ces résultats $St(0)$. Concernant le filtrage heuristique, on a testé plusieurs valeurs de K et on a choisi la plus petite valeur qui permettait d'avoir une solution de chaque instance. Le paramètre obtenu pour ces instances est $K = 7$.

Les tableaux (5.5) et (5.6) présentent les résultats de $St(0)$, $St(1)$, $St(2)$ et $St(3)$ respectivement pour les instances du paquet INST_CTE et du paquet INST_VAR. La signification des colonnes est :

- Obj** est la valeur obtenue;
- Gap** est le gap par rapport à l'optimalité $Gap = 100 \times \frac{Obj-opt}{opt}$; où Obj est la valeur obtenue et opt est la valeur optimale obtenue à l'aide du modèle $MILP_{SMEPC}$ du chapitre précédent. Si une instance n'a pas été résolu jusqu'à l'optimalité, on utilise la borne supérieure du modèle exécuté sur CPLEX sur 8 threads avec un temps limite de 3 heures (voir tableau (4.7) et (4.8)).
- #Etats** est le nombre d'états maximal générés
- CPU** est le temps d'exécution.

On exécute plusieurs fois l'algorithme **DPS_SMEPC** en activant à chaque fois les filtrages qui nous intéressent. La borne supérieure a été calculée par l'algorithme (7). On n'a pas inscrit dans le tableau les valeurs objectives de $St(2)$ et $St(1)$ car elles sont identiques à la valeur de $St(3)$. On ne calcule pas le gap de $St(1)$, $St(2)$ et $St(3)$ car on obtient la valeur optimale donc le gap vaut zéro.

On remarque que lorsqu'on ajoute des mécanismes de filtrage l'algorithme est de plus en plus rapide car les mécanismes de filtrage diminuent le nombre d'états. En comparant le nombre d'états $\#Etats$ des algorithmes, on remarque que le mécanisme de filtrage exact le plus efficace est le filtrage par estimation optimiste.

On fait les remarques suivantes :

- L'ajout du filtrage heuristique ($ST(0)$) permet d'obtenir une solution à toutes les instances et ces solutions sont proches (inférieure à 6%) de celles obtenues par CPLEX (3 heures et 8 threads). Pour 5 instances du paquet INST_VAR on obtient de meilleures solutions que la borne supérieure calculée par CPLEX;
- On constate que le temps CPU de la colonne $ST(0)$ est très faible comparé aux résultats de CPLEX. De plus l'algorithme devient plus rapide avec le filtrage heuristique qu'avec uniquement les filtrages exactes;
- C'est uniquement sur le paquet d'instances INST_VAR qu'on voit une diminution du gap lors de l'ajout du filtrage heuristique. On résout des instances qu'on ne résolvait pas avec les filtrages exactes.

- Toutes les instances du paquet INST_CTE sont résolues jusqu'à l'optimum avec les filtrages exactes. Or, ce n'est pas le cas pour le paquet d'instances INST_VAR. En effet, on n'arrive pas à résoudre 17 instances du paquet INST_VAR lorsqu'on utilise uniquement les filtrages exactes ;
- Les solutions obtenues par $ST(0)$ sont meilleures que les valeurs des heuristiques de la section précédente. Ceci est dû au fait que la meilleure valeur calculée par les heuristiques est utilisée comme borne supérieure lors du filtrage par estimation optimiste ;
- Le nombre d'états obtenus par le paquet d'instances INST_VAR est supérieur au nombre d'états du paquet d'instances INST_CTE. On fait la même remarque pour le temps de calcul.

Impact des mécanismes de filtrage du DPS_SMEPC : les instances du paquet INST_VAR

Le tableau (5.5) présente les résultats des instances du paquet INST_VAR. Lorsque la valeur vaut « - » cela veut dire que l'algorithme s'est arrêté au bout d'une heure sans trouver de solution. Le gap moyen de $St(0)$ est 0,596%. En analysant les résultats des instances du paquet INST_VAR, on remarque que le filtrage heuristique est efficace car on obtient toujours une solution de bonne qualité en moins d'une heure. Or, en ajoutant les filtrages exactes au DPS_SMEPC ($St(1)$, $St(2)$ et $St(3)$), on ne parvient pas à résoudre 56,6% des instances en moins d'une heure. De plus, en ajoutant le filtrage heuristique la méthode devient très rapide et on obtient des gaps qui sont inférieurs à 5%, ces filtrages ne dégradent que légèrement la solution. De plus avec le filtrage heuristique on obtient 6 solutions optimales en moins de 0,02 seconde. Par exemple, pour l'instance 16, on passe de 1,02 heure de temps d'exécution lorsqu'on exécute sans filtrage à 0,037 seconde lorsqu'on exécute avec tous les filtrages et on a un gap de 0,08%. Globalement le nombre d'états moyen des colonnes $St(0)$, $St(3)$, $St(2)$, $St(1)$ est respectivement 2449,133 ; 799762,7333 ; 962697,333 ; 1033360,2 et le temps d'exécution moyen en minutes est respectivement 0,96 ; 36,1048 ; 44,11 ; 44,75.

	St(0)				St(3)			St(2)		St(1)	
	Tous les filtrages exacts et heuristiques				Tous les filtrages exacts			Filtrages logiques et optimistes		Sans filtre	
num	Obj	Gap	#Etats	CPU	Obj	#Etats	CPU	#Etats	CPU	#Etats	CPU
1	131	0,00	61	0,00	131	554	0,00	4329	0,02	4604	0,03
2	155	2,65	161	0,00	151	1739	0,01	4241	0,04	4820	0,04
3	144	0,00	63	0,00	144	2213	0,03	129720	15,38	131744	15,91
4	140	0,00	321	0,01	140	2155	0,03	4748	0,06	4865	0,07
5	161	0,00	48	0,00	161	406	0,01	82005	9,62	83001	9,75
6	181	1,69	198	0,01	178	9960	0,26	13761	0,65	19072	0,84
7	222	0,00	126	0,00	222	1732	0,01	7914	0,07	14375	0,17
8	192	0,00	87	0,00	192	1607	0,02	37060	4,23	39376	4,44
9	661	2,64	286	0,04	644	263867	329,67	1410815	3910,25	1344462	4321,23
10	1149	0,88	148	0,02	1139	326094	127,00	716815	3810,17	1024128	3676,93
11	140	4,48	29	0,00	134	536	0,01	42913	6,75	43867	6,98
12	914	0,22	314	0,14	-	613249	3613,47	1020778	3856,31	1020778	3707,92
13	967	1,15	59	0,01	956	7115	0,70	1610932	3673,96	1610932	3844,40
14	1372	0,00	642	0,63	-	1550626	3602,32	1559487	3710,34	1559487	3862,77
15	1354	0,59	106	0,03	1346	261078	408,83	1067321	3799,09	1115578	4056,97
16	1292	0,08	429	0,37	-	909386	4080,64	1118045	3751,39	1118045	3684,39
17	1288	0,23	410	0,71	-	703595	3809,08	1019333	3769,64	1169832	3678,05
18	2427	2,28	390	0,44	-	872810	3701,67	764772	3664,77	1256433	3631,32
19	2218	0,23	442	1,94	-	2238483	3642,79	2010204	3654,53	2238483	3660,75
20	2657	-0,08	609	1,43	-	1137757	3646,60	918477	3839,88	1227651	3759,62
21	1412	0,14	2138	10,98	-	1207299	3671,79	846485	3611,51	1219554	3631,16
22	1350	0,60	5148	38,67	-	1759894	4088,15	1687911	3630,61	1850181	4065,63
23	2598	0,31	1284	4,92	-	1320005	3694,15	1320005	3672,38	1320005	3660,03
24	2808	0,11	1116	5,18	-	1450961	3738,81	1450961	3626,02	1450961	3660,26
25	2060	0,34	869	7,89	-	1626287	3715,09	1484155	3660,78	1484155	3639,51
26	2353	-0,13	53759	1604,02	-	1550660	3638,91	1637832	3642,65	1637832	3782,43
27	2500	1,30	696	4,26	-	1817157	3624,48	2510242	4460,81	2510242	4507,72
28	3053	-0,03	1258	11,85	-	1468315	3853,50	1468854	3727,84	1468854	4227,10
29	4390	-1,06	1070	8,65	-	1764386	4155,38	1764386	4127,95	1861070	3726,37
30	3496	-0,74	1207	17,69	-	1122956	3845,23	1166419	3758,72	1166419	3723,97

TABLE 5.5 – Les instances du paquet INST_VAR : impact des mécanismes de filtrage du DPS_SMEPC.

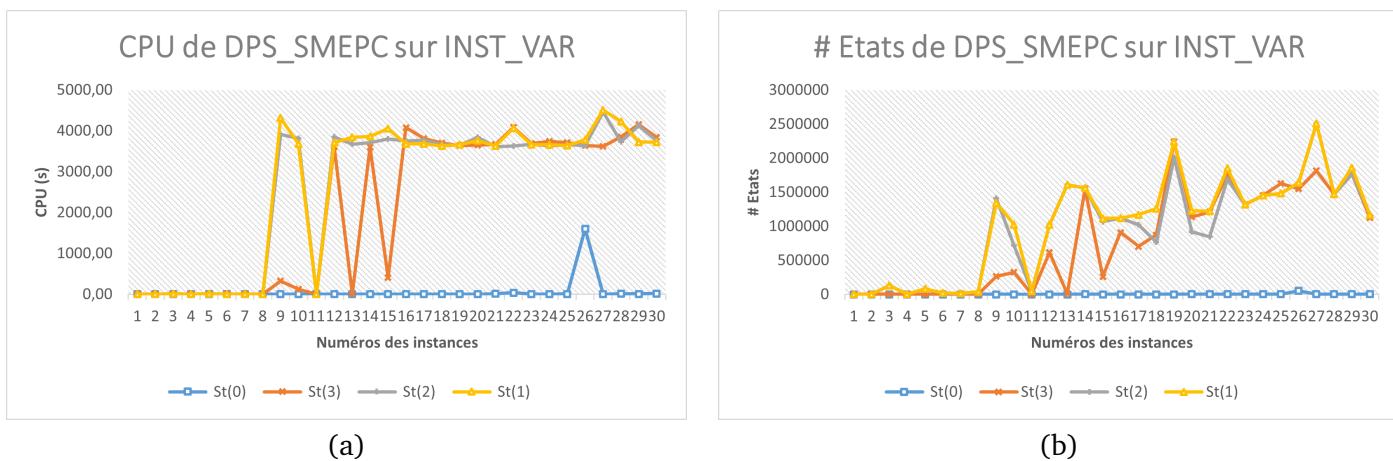


FIGURE 5.15 – Représentation graphique du tableau (5.5). (a) représente le temps CPU et (b) représente le nombre d'états de chaque instance de INST_VAR.

Impact des mécanismes de filtrage du DPS_SMEPC : les instances du paquet INST_CTE

Le tableau (5.6) présente les résultats des instances du paquet INST_CTE. Le gap moyen de $St(0)$ est 1,108%. En analysant les résultats des instances du paquet INST_CTE, on remarque que le filtrage heuristique est efficace car en ajoutant ce filtrage, la méthode devient très rapide et les gaps sont inférieurs à 7%, ces filtrages ne dégradent que légèrement la solution. De plus, avec le filtrage heuristique on obtient 60% de solutions optimales. Par exemple, pour l'instance 48, on passe de 52,85 minutes de temps d'exécution lorsqu'on exécute sans filtrage à 0,02 seconde lorsqu'on exécute avec tous les filtrages et on a un gap de 6%. Globalement le nombre d'états moyen de $St(0)$, $St(3)$, $St(2)$, $St(1)$ est respectivement 118,2 ; 13746,85 ; 191125,9 ; 216886,55 et le temps d'exécution moyen en seconde est respectivement 0,007 ; 2,448 ; 129,874 ; 316,798.

num	St(0)				St(3)				St(2)				St(1)	
	Tous les filtrages exacts et heuristiques				Tous les filtrages exacts				Filtrages logiques et optimistes				Sans filtrage	
	Obj	Gap	#Etats	CPU	Obj	#Etats	CPU	#Etats	CPU	#Etats	CPU	#Etats	#Etats	CPU
31	241	0,00	131	0,00	241	3077	0,03	10083	0,04	11422	0,06			
32	373	4,19	80	0,00	358	1074	0,01	6042	0,02	6834	0,03			
33	223	0,00	56	0,00	223	1670	0,01	11016	0,06	11504	0,07			
34	202	0,00	120	0,00	202	5631	0,06	9757	0,09	11491	0,12			
35	257	0,00	164	0,01	257	6728	0,11	20679	0,34	23413	0,51			
36	226	0,00	48	0,00	226	531	0,01	67657	2,24	68316	2,47			
37	177	0,00	46	0,00	177	602	0,01	64720	3,11	66144	3,40			
38	282	0,00	190	0,01	282	5513	0,06	11052	0,14	15478	0,25			
39	323	3,19	110	0,00	313	8724	0,08	14777	0,21	18045	0,35			
40	336	0,00	149	0,01	336	24125	0,68	314059	18,72	327525	23,93			
41	915	1,55	301	0,02	901	35121	3,99	62471	9,23	82386	15,97			
42	1440	0,00	227	0,02	1440	16635	1,28	27568	2,10	40821	4,81			
43	1623	5,66	169	0,01	1536	31508	2,77	31508	2,75	61652	10,71			
44	919	0,00	75	0,00	919	4957	0,18	117870	27,46	131961	33,55			
45	939	1,08	73	0,01	929	8930	0,82	221946	93,68	240786	124,16			
46	852	0,12	98	0,02	851	11841	1,09	388485	279,66	424646	343,15			
47	942	0,00	72	0,01	942	14009	1,45	841912	686,84	890054	1444,54			
48	1325	6,00	117	0,02	1250	64992	32,71	662529	664,03	893074	3171,79			
49	831	0,00	44	0,01	831	5497	0,49	474977	488,81	501320	744,16			
50	1084	0,37	94	0,01	1080	23772	3,13	463410	317,95	510859	411,94			

TABLE 5.6 – Les instances du paquet INST_CTE : impact des mécanismes de filtrage du DPS_SMEPC.

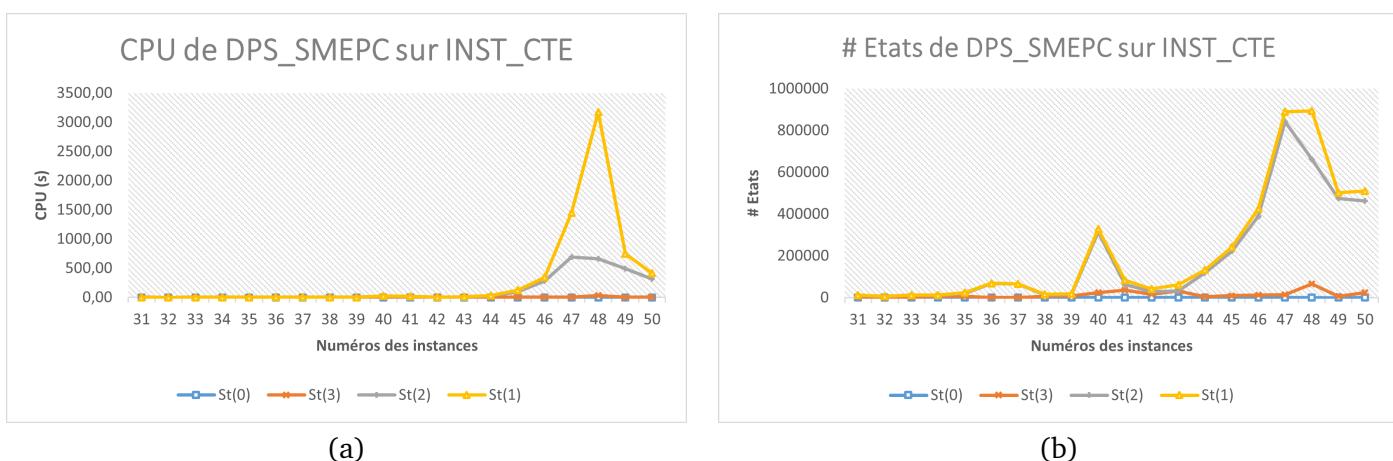


FIGURE 5.16 – Représentation graphique du tableau (5.6). (a) représente le temps CPU et (b) représente le nombre d'états de chaque instance de INST_CTE.

5.7 Conclusion

Nous avons présenté dans ce chapitre un schéma de programmation dynamique qui résout un problème d'ordonnancement qui nécessite de synchroniser les tâches effectuées par des véhicules avec un processus de production d'énergie. En raison du projet IMOBS3, nous traitons ici de la gestion synchrone, d'une part, d'une flotte de petits véhicules électriques équipés de piles à hydrogène et, d'autre part, d'une micro-usine chargée de la production locale de carburant à base d'hydrogène. Pris dans son ensemble, le problème concerne la prévision, la gestion de la sécurité et la programmation. Toutefois, comme notre objectif dans ce chapitre était de nous concentrer sur les caractéristiques algorithmiques de la synchronisation, nous nous sommes limités à ce dernier point et nous avons établit un modèle simplifié de production et de consommation d'énergie (SMEPC), limité au cas d'un véhicule devant effectuer des tâches selon un ordre préétabli tout en retournant périodiquement à la micro-usine pour faire le plein. La micro-usine a ses propres restrictions de production/stockage, et notre objectif était de synchroniser à la fois la micro-usine et le véhicule. Ce modèle est NP-Hard, et sa formulation linéaire est mal adaptée au traitement numérique. Nous l'avons abordé ici selon un paradigme purement centralisé, et nous avons présenté un schéma de programmation dynamique (DPS), qui met en œuvre la synchronisation tout en reliant les espaces temporels de la production et du véhicule. Nous avons enrichi ce DPS de mécanismes de filtrage exacts et heuristiques et il en ressort que le filtrage exact le plus efficace est le mécanisme de filtrage par estimation optimiste.

Ce DPS nous a fait énoncer un résultat PTAS (Polynomial Time Approximation Scheme) et fournir des solutions optimales. De plus, nous constatons que l'intégration complète des véhicules et de la production induit un manque de flexibilité qui rendra difficile la gestion des caractéristiques de collaboration dans les contextes de la vie réelle. Dans le prochain chapitre, nous changeons donc de paradigme et adoptons un point de vue qui consiste à aborder le **SMEPC** en émulant ces caractéristiques de collaboration : On divise notre modèle **SMEPC** en deux sous-modèles indépendants, l'un lié aux véhicules et l'autre à la micro-usine, que nous abordons tous les deux par le biais de DPS et que nous relierons entre eux par une sorte d'interaction collaborative entre le demandeur et le producteur. De nombreuses questions restent à résoudre à l'instar de comment étendre notre approche à plusieurs véhicules. Le chapitre suivant sera donc consacré à la présentation d'un schéma collaboratif pour résoudre notre problème.

5.8 Annexes

5.8.1 Démonstration du théorème 4

Cette section présente la démonstration du théorème 4. K étant fixé, le fait que l'algorithme **DPS_SMEPC(K)** soit polynomial en temps provient du fait que $TMax$, C^{Tank} et C^{Veh} sont censés être limités par des fonctions polynomiales de N et M : le nombre de triplets possibles $(s, W, Father)$ dans la liste $S(i, j)$ que nous traitons à chaque itération de la boucle principale est limité par une fonction polynomiale de N , M et la taille de codage de $TMax$, C^{Tank} et C^{Veh} .

De la même manière, étant donné ε , on voit que si K est suffisamment grand, alors l'erreur relative $(Round^*(TMax, K) - TMax)/TMax$ induite par le remplacement de $TMax$ par $Round^*(TMax, K)$ ne dépasse pas ε . Il en va de même pour les capacités C^{Veh} et C^{Tank} .

Afin de réaliser la preuve du théorème 4, nous devons maintenant envisager une solution optimale Sol^{Opt} , donnée avec sa valeur W^{Opt} . La solution Sol^{Opt} peut être associée à une séquence de dates

(i_h, j_h) , $h = 0, \dots, H \leq N + M$, une séquence d'états s_0, s_1, \dots, s_H , avec la valeur W_0, \dots, W_H , et une séquence de décisions D_0, \dots, D_{H-1} , qui induit des transitions $((i_h, j_h), s_h) \rightarrow ((i_{h+1}, j_{h+1}), s_{h+1})$, $h = 0, \dots, H - 1$, et de vérifier que si K est suffisamment grand, **DPS_SMEPC(K)** calcule une solution *Current_Sol* qui est réalisable par rapport aux valeurs seuils $(1 + \varepsilon) \times C^{Tank}$, $(1 + \varepsilon) \times C^{Veh}$ et $(1 + \varepsilon) \times TMax$ et dont la valeur de coût *Current_Value* n'est pas supérieure à $W^{Opt} + \varepsilon$. Pour ce faire, on fixe K et on prouve par induction sur h que, pour tout $h = 0, \dots, H$, Il existe dans l'ensemble $S(i_h, j_h)$ calculé par le **DPS_SMEPC(K)** quelque triplet $(s = (Z, T, V^{Tank}, V^{Veh}), W, Father)$ de sorte que, si on a $s_h = (Z_h, T_h, V_h^{Tank}, V_h^{Veh})$:

$$W \leq W_h \quad (5.4a)$$

$$Z_h = Z \quad (5.4b)$$

$$T \leq T_h \times (1 + h \times 2^{-K}/(N + M)) \quad (5.4c)$$

$$V^{Tank} \times (1 + h \times 2^{-K}/2 \times (N + M)) \leq V_h^{Tank} \leq V^{Tank} \times (1 + h \times 2^{-K}/(N + M)) \quad (5.4d)$$

$$V^{Veh} \times (1 + h \times 2^{-K}/2 \times (N + M)) \leq V_h^{Veh} \leq V^{Veh} \times (1 + h \times 2^{-K}/(N + M)) \quad (5.4e)$$

Nous supposons que c'est vrai pour un h donné et essayons d'appliquer la décision D_h à l'état s_h . Nous déduisons des équations (5.4b) et (5.4c) que l'application de D_h respectera le seuil $Round^*(TMax, K)$. A cause de l'équation (5.4d), D_h ne rendra pas la quantité d'hydrogène dans la micro-usine négative ou ne dépassera pas le $Round^*(C^{Tank}, K)$. De même, (5.4e) implique que D_h ne fera pas en sorte que la quantité d'hydrogène dans le véhicule soit négative ou dépasse le $Round^*(C^{Veh}, K)$. Il s'ensuit que cette décision va être réalisable. Nous voyons alors que l'état $s' = (Z', T', V'^{Tank}, V'^{Veh})$ résultant à la valeur temporelle (i_{h+1}, j_{h+1}) de l'application de D_h à l'état s et à la valeur W' correspondante sera tel que :

$$W' \leq W_{h+1} \quad (5.5a)$$

$$Z_{h+1} = Z' \quad (5.5b)$$

$$T' \leq T_{h+1} \times (1 + h \times 2^{-K}/(N + M)) \quad (5.5c)$$

$$V'^{Tank} \times (1 + h \times 2^{-K}/2 \times (N + M)) \leq V_{h+1}^{Tank} \leq V'^{Tank} \times (1 + h \times 2^{-K}/(N + M)) \quad (5.5d)$$

$$V'^{Veh} \times (1 + h \times 2^{-K}/2 \times (N + M)) \leq V_{h+1}^{Veh} \leq V'^{Veh} \times (1 + h \times 2^{-K}/(N + M)) \quad (5.5e)$$

Mais (5.5b, 5.5c, 5.5d, 5.5e) combiné avec les mécanismes de propagation liées aux erreurs relatives impliquent que si $(s^*, W^*, Father^*)$ est l'élément équivalent à (s', W') modulo les $K + L$ plus grands bits qui restent dans $S(i_{h+1}, j_{h+1})$ du **DPS_SMEPC(K)**, alors nous avons :

$$W^* \leq W_{h+1} \quad (5.6a)$$

$$Z_{h+1} = Z^* \quad (5.6b)$$

$$T^* \leq T_{h+1} \times (1 + (h+1) \times 2^{-K}/(N+M)) \quad (5.6c)$$

$$V^{Tank*} \times (1 + (h+1) \times 2^{-K}/2 \times (N+M)) \leq V_{h+1}^{Tank} \leq V^{Tank*} \times (1 + (h+1) \times 2^{-K}/(N+M)) \quad (5.6d)$$

$$V^{Veh*} \times (1 + (h+1) \times 2^{-K}/2 \times (N+M)) \leq V_{h+1}^{Veh} \leq V^{Veh*} \times (1 + (h+1) \times 2^{-K}/(N+M)) \quad (5.6e)$$

Nous déduisons que les équations (5.4b, 5.4c, 5.4d, 5.4e) sont vraies pour tout $h = 0, \dots, H$. Cela nous permet de choisir K de telle manière que $2K$ est plus grand que $(N+M+1)/\varepsilon$.

CHAPITRE 6

SCHÉMA COLLABORATIF POUR RÉSOUTRE SMEPC

Sommaire

6.1 Motivations	160
6.2 Le problème du véhicule : <i>Vehicle-Driver</i> (VD)	160
6.2.1 Le modèle VD	161
6.2.2 Complexité du modèle VD avec L_j fixe	163
6.2.3 Complexité du modèle VD avec L_j variable	163
6.3 Le problème de production : <i>Production-Manager</i> (PM)	163
6.3.1 Le modèle PM	163
6.3.2 Complexité du modèle PM	165
6.4 L'algorithme de programmation dynamique DPS_VD	165
6.4.1 Espace temps et états au sens de la programmation dynamique	166
Etat initial et Etats finaux	166
6.4.2 Variable de décision	166
6.4.3 Pré-conditions, transitions et coûts liés aux décisions et aux transitions	167
6.4.4 Mise en œuvre algorithmique du DPS_VD	167
Equations de Bellman	167
Stratégie d'exploration	167
Algorithme DPS_VD	167
6.4.5 Calcul de la stratégie de recharge primaire et de la stratégie de recharge réduite	168
6.4.6 Calcul des bornes inférieures du temps et de l'énergie du DPS_SMEPC	171
6.5 L'algorithme de programmation dynamique DPS_PM	172
6.5.1 Espace temps et états au sens de la programmation dynamique	172
Etat initial et Etats finaux	173
6.5.2 Variables de décision	173
6.5.3 Pré-conditions, transitions et coûts liés aux décisions et aux transitions	173
6.5.4 Mise en œuvre algorithmique du DPS_PM	174
Equations de Bellman	175
Stratégie d'exploration	175
Algorithme DPS_PM	175
6.5.5 Mécanismes de filtrage du DPS_PM	175

Mécanismes de filtrage par arrondi	178
Mécanismes de filtrage par relations de dominance	178
Mécanismes de filtrage logique	178
Mécanismes de filtrage par estimation optimiste	178
Mécanismes de filtrage heuristique	181
6.6 Collaboration Demandeur/Producteur pour SMEPC : le schéma Pipe-line VD_PM	181
6.7 Expérimentations numériques	185
6.7.1 Objectifs et contexte technique	185
6.7.2 Instances	185
6.7.3 Comparaison de l'heuristique Pipe-line VD_PM et du DPS_SMEPC	187
Comparaison de l'heuristique Pipe-line VD_PM et du DPS_SMEPC : les instances du paquet INST_VAR	189
Comparaison de l'heuristique Pipe-line VD_PM et du DPS_SMEPC : les instances du paquet INST_CTE	191
6.7.4 Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC	194
Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC : les instances du paquet INST_VAR	196
Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC : les instances du paquet INST_CTE	197
6.8 Conclusion	198
6.9 Annexes	198
6.9.1 Démonstration du théorème 5	198
6.9.2 Démonstration du théorème 6	201

6.1 Motivations

La synchronisation efficace de processus hétérogènes reste une question difficile lorsqu'il s'agit de programmation et d'acheminement (voir [35]). Elle se pose par exemple dans la gestion des systèmes de partage de véhicules, des drones et des camions dans le contexte de la logistique urbaine, et des processus d'assemblage industriel. Les modèles de programmation linéaire en nombres entiers sont faussés par de grands écarts induits par l'assouplissement de la contrainte d'intégrité (Méthode du Big M). De même, il est difficile de concevoir des systèmes du type *Branch and Bound* en raison de l'absence d'un système efficace de génération de bornes. En outre, les exigences de synchronisation augmentent l'impact de l'incertitude et mettent en jeu la robustesse, ce qui rend l'efficacité des heuristiques globales difficile à vérifier. Une façon de résoudre ces problèmes est d'introduire de la flexibilité et de la modularité dans la conception des algorithmes et de s'appuyer sur des schémas de décomposition adaptés afin d'émuler les mécanismes d'interaction qui permettent à des acteurs distincts de faire fonctionner un processus complexe de manière décentralisée.

L'algorithme de programmation dynamique décrit au chapitre précédent implique un trop grand nombre d'états car il devient vite exponentiel en fonction de la taille de l'instance, ce qui nécessite une très grande mémoire. De plus, on a du mal à apporter une modification à ce programme dynamique car il est « *lourd* ».

Nous allons expliquer ici la manière dont le modèle **SMEPC** présenté auparavant peut être décomposé de façon à reproduire comment les décisions vont être prises dans le cas (réaliste) où la décision est prise en collaboration. Ceci signifie que le directeur de production et le conducteur du véhicule sont des acteurs indépendants qui doivent interagir. L'idée principale à l'œuvre est qu'un conducteur de véhicule décentralisé a pour comportement naturel de se déplacer comme s'il était sûr d'avoir assez de carburant chaque fois qu'il se rend à la micro-usine, il s'adapte alors au contexte réel en attendant d'être rechargé retardant ainsi certains déplacements. Le comportement induit du directeur de production consistera à assurer le plus possible la demande, en faisant un compromis entre la qualité de service et le coût de production.

Dans ce chapitre, on décrit une décomposition du problème **SMEPC** à l'aide de deux modèles qui interagiront : le modèle nommé **Vehicle-Driver (VD)** présenté à la **section 6.2** et le modèle nommé **Production Manager (PM)** présenté à la **section 6.3**. On étudie **VD** et **PM**, du point de vue algorithmique respectivement à la **section 6.4** et à la **section 6.5**. Nous y allons aussi décrire les schémas de programmation dynamique **DPS_VD** et **DPS_PM** que nous proposons pour résoudre respectivement le problème **VD** et le problème **PM**. Pour diminuer le nombre d'états généré par l'algorithme **DPS_PM**, on va comme au chapitre précédent définir des règles de filtrages exactes et heuristiques. On conçoit un schéma algorithmique collaboratif qu'on nomme **Pipe-line VD_PM** qui met en œuvre la décomposition Demandeur/Producteur à la **section 6.6**. On fait des expérimentations numériques à la **section 6.7**, qui visent à évaluer la qualité des solutions obtenus par le schéma collaboratif.

6.2 Le problème du véhicule : *Vehicle-Driver (VD)*

Dans cette section, on va commencer par présenter le modèle **VD** puis on donnera les résultats de complexité du modèle **VD**.

6.2.1 Le modèle VD

Partant du problème global désigné par **SMEPC**, on décide de faire abstraction de l'aspect planification de la production en supposant qu'il y a suffisamment d'hydrogène à la micro-usine pour alimenter le véhicule, peu importe le nombre de fois qu'il ira se recharger. Il se pose le problème de savoir à quelle date (entre quelles stations j et $j + 1$) le véhicule se rechargeera et en quelle quantité d'hydrogène ? Afin d'apporter des éléments de réponse à ces questions, on décide de concevoir un algorithme DPS qui fonctionne en *Backward*. Dans cette section, pour présenter le modèle **VD**, on va d'abord lister les entrées du modèle **VD**, ensuite, on va présenter ce qu'est une solution du modèle **VD** et enfin, on va présenter la fonction objectif du modèle **VD**.

Dans le chapitre précédent, on a présenté toutes les entrées du modèle **SMEPC** (voir le tableau (4.1)). Le tableau (6.1) résume les entrées du problème **Vehicle-Driver (VD)** qui est un sous-problème du problème **SMEPC**.

Noms	Significations
M	Nombre de stations (Dépôt exclut)
$\Gamma = (Depot = 0, 1, \dots, M, Depot = M + 1)$	Tournée fixe du véhicule (sans les recharges)
$TMax$	Le délai maximal pour que le véhicule puisse effectuer sa tournée
C^{Veh}	Capacité du réservoir d'hydrogène du véhicule
E_0	Charge initiale d'hydrogène du véhicule
p	Durée de la recharge du véhicule
Pour $j = 0, \dots, M, t_j$	Temps nécessaire pour aller de la station j à la station $j + 1$
Pour $j = 0, \dots, M, d_j$	Temps nécessaire pour aller de la station j à la micro-usine
Pour $j = 0, \dots, M, d_j^*$	Temps nécessaire pour aller de la micro-usine à la station j
Pour $j = 0, \dots, M, e_j$	Energie nécessaire pour aller de la station j à la station $j + 1$
Pour $j = 0, \dots, M, \varepsilon_j$	Energie nécessaire pour aller de la station j à la micro-usine
Pour $j = 0, \dots, M, \varepsilon_j^*$	Energie nécessaire pour aller de la micro-usine à la station j

TABLE 6.1 – Entrées correspondantes au problème **VD**.

On oublie ici les restrictions liées à la production d'hydrogène : nous faisons comme si la micro-usine était capable de fournir, à tout moment, au véhicule autant d'énergie qu'il en a besoin. Notre objectif est alors de fixer la stratégie de recharge du véhicule, c'est-à-dire le vecteur de valeurs booléennes $x = (x_j, j = 0, \dots, M)$ et le vecteur de charge $L = (L_j, j = 0, \dots, M)$ du modèle **SMEPC**, qui nous indiquent respectivement entre quelles stations j et $j + 1$ le véhicule se rechargeera, et de quelle quantité d'hydrogène. Les variables $T = (T_j, j = 0, \dots, M + 1)$, $T^* = (T_j^*, j = 0, \dots, M + 1)$ et $V^{Veh} = (V_j^{Veh}, j = 0, \dots, M + 1)$ peuvent être considérées comme des variables auxiliaires, dont les valeurs dérivent de x et L . Nous faisons les hypothèses suivantes dans le modèle **VD** :

- Le véhicule n'attend jamais : il se recharge en carburant lorsqu'il arrive à la micro-usine, et il garde entre-temps sa vitesse maximale ;
- A chaque fois que le véhicule se recharge, (sauf à la dernière recharge) il fait le plein de son réservoir. A la dernière recharge, il se recharge de telle manière qu'il retourne à la micro-usine

avec une charge finale au moins égale à E_0 .

De ces constats, on peut dire que les variables T , T^* et V^{Veh} signifient :

- $\square T = (T_j, j = 0, \dots, M + 1)$, la valeur entière non négative T_j représente la date d'arrivée du véhicule à la station j ;
- $\square T^* = (T_j^*, j = 0, \dots, M + 1)$, la valeur entière non négative T_j^* est la date à laquelle le véhicule commence à se recharger en hydrogène entre la station j et la station $j + 1$ si $x_j = 1$;
- $\square V^{Veh} = (V_j^{Veh}, j = 0, \dots, M + 1)$, la valeur entière non négative V_j^{Veh} représente la quantité d'hydrogène dans le réservoir du véhicule lorsqu'il arrive à la station j .

Les contraintes du véhicule sont les contraintes (4.2a), (4.2b), (4.2c), (4.3a), (4.3c), (4.3d) du modèle SMEPC. Il reste à préciser le critère de performance. Bien entendu, il doit inclure le terme $\alpha \times T_{M+1}$. Mais il doit également comporter un élément qui reflète le coût économique de la **stratégie de recharge optimale** (x, L). Comme nous ne savons pas à l'avance quelle sera la **stratégie de production**, nous introduisons un coefficient de coût auxiliaire β et considérons que le coût économique de la **stratégie de recharge optimale** (x, L) est la quantité $\beta \times (\sum_j L_j \times x_j)$. Ce coefficient va être un élément clé de l'interaction entre le véhicule (demandeur) et la micro-usine (producteur).

Le modèle du conducteur du véhicule (VD) calcule la **stratégie de recharge optimale** (x, L), et déduit les variables auxiliaires $T = (T_j, j = 0, \dots, M + 1)$, $T^* = (T_j^*, j = 0, \dots, M + 1)$ et $V^{Veh} = (V_j^{Veh}, j = 0, \dots, M + 1)$, de telle sorte que :

- \square Les contraintes imposées au véhicule dans le modèle SMEPC soient satisfaites ;
- \square La quantité $\alpha \times T_{M+1} + \beta \times (\sum_j L_j \times x_j)$ soit la plus petite possible.

Exemple 2 Le tableau (6.2) donne les distances t_j , d_j , d_j^* , et les énergies e_j , ε_j , ε_j^* d'une instance du modèle VD. Les autres données de l'instance sont : $M = 4$, $\Gamma = (0, 1, 2, 3, 4, 5 = 0)$, $TMax = 60$, $C^{Veh} = 24$, $E_0 = 21$, $p = 4$. La solution de cette instance est donnée par le tableau (6.3). Le véhicule se recharge deux fois. Le véhicule se recharge entre la station 3 et la station 4 puis entre la station 4 et la station 5 = 0 d'une quantité d'hydrogène respectivement de 12 et 22.

j	0	1	2	3	4	5=0
t_j	4	5	7	6	5	*
$d_j = d_j^*$	1	4	6	3	5	1
e_j	5	6	7	7	7	*
$\varepsilon_j = \varepsilon_j^*$	1	4	8	3	6	1

TABLE 6.2 – Une instance du modèle. VD

j	0	1	2	3	4
x_j	0	0	0	1	1
L_j	0	0	0	12	22

TABLE 6.3 – La stratégie de recharge optimale de l'instance du modèle VD du tableau (6.2).

Exemple 3 Le tableau (6.4) donne les distances t_j , d_j , d_j^* , et les énergies e_j , ε_j , ε_j^* d'une instance du modèle VD. Les autres données de l'instance sont : $M = 10$, $\Gamma = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 = 0)$, $TMax = 150$, $C^{Veh} = 35$, $E_0 = 5$, $p = 5$. La solution de cette instance est donnée par le tableau (6.5). Le véhicule se recharge quatre fois. Le véhicule se recharge d'abord, entre la station 0 et la station 1, ensuite,

j	0	1	2	3	4	5	6	7	8	9	10	11=0
t_j	12	10	3	3	3	8	10	9	5	6	6	*
$d_j = d_j^*$	1	11	1	3	3	3	10	8	6	7	7	1
e_j	16	14	3	3	3	10	10	12	5	6	6	*
$\varepsilon_j = \varepsilon_j^*$	1	15	1	4	3	4	14	8	8	9	7	1

TABLE 6.4 – Une instance du modèle VD

entre la station 2 et la station 3, ensuite, entre la station 3 et la station 4 et enfin entre la station 7 et la station 8 d'une quantité d'hydrogène respectivement de 26, 8, 34 et 28.

j	0	1	2	3	4	5	6	7	8	9	10
x_j	1	0	1	1	0	0	0	1	0	0	0
L_j	26	0	8	34	0	0	0	28	0	0	0

TABLE 6.5 – La stratégie de recharge optimale de l'instance du modèle VD du tableau (6.4).

6.2.2 Complexité du modèle VD avec L_j fixe

Théorème 5 Si nous imposons que tout L_j est soit nul ou égal à une constante R , alors VD devient NP-Hard.

La démonstration de ce théorème se trouve en annexe à la section 6.9.1.

6.2.3 Complexité du modèle VD avec L_j variable

Théorème 6 VD peut être résolu en temps polynomial.

La démonstration de ce théorème se trouve en annexe à la section 6.9.2.

6.3 Le problème de production : Production-Manager (PM)

Dans cette section, on commencera par décrire le modèle PM, puis on présentera les résultats de complexité du modèle PM.

6.3.1 Le modèle PM

Dans le chapitre précédent, on a présenté les entrées du modèle SMEPC. Le tableau (6.6) résume les entrées du problème PM qui est un sous-problème du problème SMEPC.

Noms	Significations
C^{Tank}	Capacité de la citerne d'hydrogène
N	Nombre de périodes de production
p	Durée en unités de temps d'une période de production
H_0	Charge initiale de la citerne d'hydrogène
$Cost^F$	Coût d'activation de la micro-usine
Pour $i = 0, \dots, N - 1$, $P_i = [p \times i, p \times (i + 1)[$	Intervalle de temps correspondant à une période de production
Pour $i = 0, \dots, N - 1$, R_i	Rendement de production lié à la période i
Pour $i = 0, \dots, N - 1$, $Cost_i^V$	Coût de production lié à la période i

TABLE 6.6 – Entrées correspondantes au problème PM.

Comme le directeur de production est censé s'adapter à la demande du véhicule, nous supposons ici qu'il reçoit la demande de carburant du conducteur du véhicule, qui découle d'une stratégie de recharge. En fait, une **stratégie de recharge optimale** (x, L) nous fournit un nombre Q d'opérations de recharge effectuées par le véhicule, avec les charges d'hydrogène μ_q , $q = 1, \dots, Q$, et les dates optimales auxquelles ces opérations de recharge ont lieu. Mais on comprend que ces dates vont être le problème pour un accord entre le véhicule et la micro-usine. Afin d'apporter de la flexibilité à cet accord, nous utiliserons le vecteur x pour calculer des bornes inférieures m_1, \dots, m_Q et des bornes supérieures M_1, \dots, M_Q pour les périodes où les opérations de recharge ont lieu, ainsi que les retards minimaux (décalages) B_1, \dots, B_Q entre deux de ces périodes consécutives, en raison du trajet que le véhicule doit effectuer entre deux opérations de recharge consécutives.

Il s'ensuit que les données d'entrées du problème PM sont alors :

- Les coûts de production $Cost^F$, $Cost_i^V$, avec $i = 0, \dots, N - 1$, la capacité C^{Tank} de la micro-usine et le rendement de production R_i , avec $i = 0, \dots, N - 1$;
- La valeur initiale H_0 du réservoir de la micro-usine ;
- Un nombre Q d'opérations de recharge en carburant effectuées par le véhicule ; Ces opérations sont étiquetées $q = 1, \dots, Q$, et sont censées se dérouler selon cet ordre ;
- Les charges μ_q , $q = 1, \dots, Q$, d'hydrogène qui sont chargées par le véhicule lors de toute opération de recharge $q = 1, \dots, Q$;
- Les bornes inférieures m_1, \dots, m_Q et les bornes supérieures M_1, \dots, M_Q pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en carburant $q = 1, \dots, Q$ auront lieu, ainsi que les **coefficients de décalage temporel** B_1, \dots, B_Q qui expriment les contraintes : Pour tout $q = 1, \dots, Q - 1$, $i_{q+1} \geq i_q + B_q$.
- Un coefficient λ utilisé dans la fonction objectif pour convertir le temps en coût économique.

Notre objectif est de programmer l'activité de la micro-usine qu'on désigne ici par **stratégie de production**, c'est-à-dire de décider, pour chaque période $i = 0, \dots, N - 1$, si la micro-usine produit ($z_i = 1$) et si la micro-usine est activée ($y_i = 1$) de cette manière :

- Le véhicule peut faire le plein aux périodes i_1, \dots, i_Q d'une manière compatible avec les contraintes de décalage et de fenêtre temporelle ci-dessus ;
- La micro-usine se termine avec une charge d'hydrogène au moins égale à la quantité H_0 avec laquelle elle a commencé ;

- La quantité $\lambda \times i_Q + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i)$ est la plus petite possible.

Plus précisément, on veut calculer les vecteurs booléens z et δ avec une indexation sur $i = 0, \dots, N - 1$ avec la même signification qu'à la section 4.4, de telle sorte que :

- Les contraintes (4.4a), (4.4b), (4.4c), (4.4d), (4.4e), (??), (4.5a), (4.5b), (4.5c), (4.5d) de production de la section 4.4 soient satisfaites (le vecteur y est un vecteur auxiliaire) ;
- Les valeurs i lorsque $\delta_i = 1$ correspondent à Q périodes i_1, \dots, i_Q qui sont en accord avec les bornes inférieures m_1, \dots, m_Q , les bornes supérieures M_1, \dots, M_Q et les coefficients de décalage B_1, \dots, B_Q ;
- Les valeurs L_i de la section 4.4 du modèle **SMEPC** correspondent aux valeurs L_q , $q = 1, \dots, Q$.

Quant au critère de performance, il doit clairement impliquer le coût économique $\sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i)$. Mais il doit également contenir une composante qui reflète le rôle du terme $\alpha \times T_{M+1}$ de la fonction objectif du modèle global **SMEPC**. Il s'ensuit que la stratégie de production (z, δ) doit viser à minimiser la quantité $\alpha \times p \times i_Q + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i)$. Le modèle de gestion de la production se présente comme suit :

PM : Modèle de gestionnaire de production : Calculez la stratégie de production (z, δ) , avec les variables auxiliaires $y = (y_i, i = 0, \dots, N - 1)$, $V^{Tank} = (V_i^{Tank}, i = 0, \dots, N - 1)$ et $L = (L_i, i = 0, \dots, N - 1)$, de telle sorte que :

- Les contraintes de production de la section 4.4 sont satisfaites ;
- $\sum_i \delta_i = Q$: les indices i tels que $\delta_i = 1$ peuvent être étiquetés $i_1 < \dots < i_Q$ de telle sorte que :
 - Pour tout q , $\mu_q = L_{iq}^*$;
 - Pour tout q : $m_q \leq i_q \leq M_q$;
 - Pour tout $q \leq Q - 1$, $i_{q+1} - i_q \leq B_q$;
 - La quantité $\alpha \times p \times i_Q + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i)$ est la plus petite possible

6.3.2 Complexité du modèle PM

Notez que **PM** est un problème *NP-Hard*, car il peut clairement être considéré comme une extension du problème du sac à dos avec les dates de production. En fait, il suffit de fixer $Q = 1$, $m_1 = M_1 = N - 1$ pour faire coïncider le problème **PM** avec le problème du sac à dos.

6.4 L'algorithme de programmation dynamique DPS_VD

Nous allons décrire ici l'algorithme de programmation dynamique **DPS_VD** que nous proposons pour résoudre le problème **VD**.

Nous supposons que la micro-usine est capable de fournir au véhicule à tout moment autant d'hydrogène qu'il en a besoin. Notre objectif est alors de décider de la stratégie de recharge optimale du véhicule, représenté par les vecteurs x et L . Le vecteur de booléens $x = (x_j, j = 0, \dots, M)$, $x_j \in \{0, 1\}$ indique pour chaque station j , que le véhicule va se recharger en hydrogène entre j et $j + 1$. Le vecteur $L = (L_j, j = 0, \dots, M)$ représente la quantité d'hydrogène rechargée à chaque recharge du véhicule. On calcule les vecteurs x et L en minimisant une certaine quantité : $\alpha \times T_{M+1} + \beta \times (\sum_j L_j \times x_j)$, où T_{M+1} est la date de fin de la tournée du véhicule, et α, β sont des coefficients non négatifs. Nous sommes tenus de finir la tournée avec au moins la même charge initiale E_0 qu'au départ.

Nous pouvons concevoir la stratégie de recharge optimale du véhicule de telle sorte que chaque fois qu'il arrive à la micro-usine pour se recharger, il reçoive exactement la quantité d'hydrogène dont il a besoin pour continuer jusqu'à la prochaine opération de recharge. Cela signifie que nous pouvons faire comme si le véhicule était arrivé à la micro-usine avec un réservoir vide, sauf dans le cas de la première opération de recharge, où nous devons tenir compte de la charge initiale E_0 . En fait, nous pouvons également inclure cette première opération de recharge dans notre schéma et reconstituer la véritable première opération de recharge. Pour cela on doit tenir compte d'une partie de l'hydrogène initiale E_0 qui n'a pas été utilisée par le véhicule à son arrivée à la micro-usine pour effectuer sa première recharge.

En fait, nous voulons effectuer notre recherche d'une **stratégie de recharge optimale** tout en obtenant en même temps des bornes inférieures pour la consommation d'énergie et la consommation de temps qui sont nécessaires pour réaliser notre tournée en partant d'une station j , avec une charge initiale V^{Veh} . Pour ce faire, nous allons concevoir un DPS (temps polynomial) qui fonctionne en *Backward*, impliquant les espaces de temps, d'état et de décisions (au sens de la programmation dynamique). Nous traitons le problème **VD** en calculant le plus court chemin dans le sous-graphe *Useful VD_Subgraph* selon un processus de Bellman en *Backward*, ce qui nous conduit au schéma algorithmique de programmation dynamique **DPS_VD** que nous décriront par la suite.

6.4.1 Espace temps et états au sens de la programmation dynamique

L'espace temps du schéma de programmation dynamique **DPS_VD** est l'ensemble $J = \{0, 1, \dots, M, M+1\}$. Nous allons parcourir l'espace temps à l'envers (en *Backward*) c'est-à-dire de la station particulière $M+1$ à la station particulière 0. Cet espace temps est indexé sur le nombre de stations y compris le dépôt.

Un état s associé au temps $j \in J$ au sens de la programmation dynamique est un couple $s = (T, V^{Veh})$ dans lequel T est le temps nécessaire au véhicule pour aller de la station j au dépôt final $M+1$. Et, V^{Veh} est la quantité d'hydrogène dans le réservoir du véhicule à la station j . Un tel état sera donné avec une valeur $W^{Veh} = \alpha \times T + \beta \times U$, où U est l'énergie qui sera gaspillée par le véhicule avant la fin de sa tournée. Toutes les paires (j, V^{Veh}) doivent appartenir à l'ensemble de noeuds X du sous-graphe *Useful VD_Subgraph*.

Etat initial et Etats finaux

L'état initial (dans le sens inverse, c'est-à-dire lié à $M+1$) sera $s^{Start} = (0, E_0)$; Les états finaux (dans le sens inverse, c'est-à-dire lié à 0) devraient être n'importe quel état $s^{End} = (T \leq TMax, V^{Veh} \leq E_0)$.

6.4.2 Variable de décision

Une décision au temps programmation dynamique j est un nombre binaire $x \in \{0, 1\}$: $x = 0$ signifie un déplacement direct du véhicule de j à $j+1$ sans recharge, tandis que $x = 1$ signifie qu'un détour de recharge vers la micro-usine doit être effectué par le véhicule avant d'atteindre $j+1$.

6.4.3 Pré-conditions, transitions et coûts liés aux décisions et aux transitions

On présente ici les transitions en *Backward* à partir de tout état s_1 au temps $j + 1$ induites par une décision x prise au temps $j \geq 0$. On a une variable de décision booléenne, les pré-conditions, transitions et coûts liés aux décisions en backward et aux transitions qui en découlent se présentent comme suit :

- Si $x = 0$, alors nous déduisons l'état $s = (T, V^{Veh})$ associé à j de l'état $s_1 = (T_1, V_1^{Veh})$ associé à $j + 1$ en fixant : $T = T_1 + t_j$; $V^{Veh} = V_1^{Veh} + e_j$; Son coût est de $\alpha \times t_j + \beta \times e_j$; Bien sûr, nous devrions avoir $T \leq TMax$ et $V^{Veh} \leq C^{Veh}$;
- Si $x = 1$, alors nous déduisons l'état $s = (T, V^{Veh})$ associé à j de l'état $s_1 = (T_1, V_1^{Veh})$ associé à $j + 1$ en fixant : $T = d_j + p + d_{j+1}^* + T_1$; $V^{Veh} = \varepsilon_j$; Son coût est $\alpha \times (d_j + p + d_{j+1}^*) + \beta \times (\varepsilon_j + \varepsilon_{j+1}^*)$; Bien sûr, nous devrions avoir $T \leq TMax$ et $V^{Veh} \leq C^{Veh}$.

6.4.4 Mise en œuvre algorithmique du DPS_VD

Nous utilisons un vecteur $STATE = (STATE_j, j = 0, \dots, M+1)$, avec indexation sur $0, 1, \dots, M+1$. $STATE_j$ est la liste des états liés à j . Nous considérons (codage le plus simple) que $STATE_j$ est un vecteur avec indexation sur le nombre maximal d'états $SMax$ que nous acceptons pour j . Il en résulte que $STATE_{j,q}$, $q = 1, \dots, SMax$, est un quadruplet $(s, W^{Veh}, x, Next)$, où $Next$ est une valeur d'index dans $1, \dots, SMax$ qui désigne l'index du père. Nous utilisons un vecteur supplémentaire $NState$ qui nous fournit, pour tout j , le nombre $NState_j$ d'états associés à j .

Equations de Bellman

Il est clair que dans le cas $j \geq 0$, et pour tout état qui y est lié s de valeur W^{Veh} , nous devrions atteindre : $W^{Veh} = \inf_{x \in \{0,1\}, x \text{ possible}} (W_1 + \text{coût de la transition induite par } x)$, où W_1 est la valeur de W^{Veh} liée à s_1 .

Lorsqu'on veut ajouter un nouvel état s^* dans le vecteur $STATE_j$ on a deux possibilités :

1. L'état s^* n'existe pas dans $STATE_j$ auquel cas on l'ajoute à $STATE_j$ en incrémentant le nombre d'états $NState_j$ de la liste.
2. L'état s^* existe déjà dans $STATE_j$ dans ce cas on applique la procédure de Bellman pour garder le meilleure état. Dans notre cas, on garde l'état qui a la plus petite valeur.

L'algorithme (8) décrit la procédure de Bellman utilisée pour ajouter un nouvel état $(s^*, W^{Veh*}, x^*, Next^*)$ dans le vecteur $STATE$.

Stratégie d'exploration

La stratégie de parcours du temps au sens de la programmation dynamique utilisé ici est la stratégie en *Backward*.

Algorithme DPS_VD

Les entrées de l'algorithme DPS_VD sont $M, TMax, E_0, C^{Veh}, e, \varepsilon, \varepsilon^*, t, d$ et d^* , α, β et, les sorties sont $STATE$ et $NState$.

Algorithme 8 Veh_Bellman_Update

Entrées: $STATE, NState, j, s^*, W^{Veh*}, x^*, Next^*$ **Sorties:** $STATE, NState$

Initialisation :

- 1: $Notstop;$
- 2: $q1 \leftarrow 1;$

Boucle principale :

- 3: **Tant que** ($q1 \leq NState_{j-1} \wedge (NotStop)$) **faire**
 - 4: **Si** $STATE_{j-1,q1}.s = s^*$ **alors**
 - 5: $Stop;$
 - 6: **sinon**
 - 7: $q1 \leftarrow q1 + 1;$
 - 8: **Fin si**
 - 9: **Fin tant que**
 - 10: **Si** $NotStop$ **alors**
 - 11: $NState_{j-1} \leftarrow NState_{j-1} + 1;$
 - 12: $STATE_{j-1,NState_{j-1}} \leftarrow (s^*, W^{Veh*}, x^*, Next^*);$
 - 13: **sinon**
 - 14: $WAux \leftarrow STATE_{j-1,q1}.W^{Veh};$
 - 15: **Si** $W^{Veh*} < WAux$ **alors**
 - 16: $STATE_{j-1,q1} \leftarrow (s^*, W^{Veh*}, x^*, Next^*);$
 - 17: **Fin si**
 - 18: **Fin si**
-

Nous suivons le théorème 6 et la construction du sous-graphe *Useful VD_Subgraph*, et construisons donc notre processus DPS_VD selon une stratégie en *Backward*. Le schéma de programmation dynamique **DPS_VD** qui fonctionne en *Backward* est développé par l'algorithme (9).

Nous allons maintenant voir comment extraire du vecteur *STATE* les solutions des instances du problème **VD** traitées avec l'algorithme **DPS_VD**, on désignera ces solutions par **stratégies de recharge primaire** et **stratégie de recharge réduite**.

6.4.5 Calcul de la stratégie de recharge primaire et de la stratégie de recharge réduite

Par la suite, on va décrire comment est extraite la stratégie de recharge, une fois que le vecteur *STATE* ait été calculé. Cette stratégie de recharge est appelée **stratégie de recharge primaire**. Calculer la stratégie de recharge primaire consiste à prendre l'état final à l'envers ($j = 0, V^{Veh} \leq E_0$), avec une valeur T non supérieure à $TMax$, ce qui nous donne la valeur $W^{Veh} - \beta \times E_0$ la plus faible, et à suivre les stations $j = 0, \dots, M$ selon les décisions x . Supposons que *STATE* ait été calculé. Nous désignons par q_0 l'indice tel que $STATE_{0,q_0}.s = (T, V^{Veh})$ satisfait $T \leq TMax$ et $V^{Veh} \leq E_0$, et induit la plus petite valeur de $STATE_{0,q_0}.W^{Veh}$. La **stratégie de recharge primaire** est un vecteur noté $LOAD = (LOAD_q, q = 0, \dots, M)$, ainsi qu'un nombre Q , ayant la signification suivante :

1. Q est le nombre d'opérations de recharge en carburant effectuées par le véhicule ;
2. Pour $q = 1, \dots, Q$, $LOAD_q$ nous fournit un quintuplet $(St, L, TInf, TSup, \Delta)$ tel que :
 - St_q est la station j tel que la q^{ieme} opération de recharge est effectuée entre la station j et la station $j + 1$;
 - L_q est la quantité d'hydrogène qui est rechargée lors de la q^{ieme} opération de recharge ;

Algorithme 9 DPS_VD**Entrées:** $M, TMax, E_0, C^{Veh}, e, \varepsilon, \varepsilon^*, t, d, d^*, \alpha, \beta$ **Sorties:** $STATE, NState$ **Initialisation :**

- 1: $j \leftarrow M + 1;$
- 2: $STATE_{M+1,1} \leftarrow ((0, E_0), 0, 0, 0); /* STATE_{M+1} ne contient qu'un unique état;$
- 3: $NState_{M+1} \leftarrow 1;$

Boucle principale :

- 4: **pour** $j = M + 1$ à 1 **faire**
- 5: **pour** $q = 1$ à $NState_j$ **faire**
- 6: Soit $STATE_{j,q} = ((T, V^{Veh}), W^{Veh}, x, Next)$
- 7: $V1 \leftarrow V^{Veh} + e_{j-1}; T1 \leftarrow T + t_{j-1}; s1 \leftarrow (T1, V1);$
- 8: **Si** $(V1 \leq C^{Veh}) \wedge (T1 \leq TMax)$ **alors**
- 9: $W1 \leftarrow \alpha \times t_{j-1} + \beta \times e_{j-1} + W^{Veh}; x1 \leftarrow 0; Next1 \leftarrow q;$
- 10: **Veh_Bellman_Update**($STATE, NState, j, s1, W1, x1, Next1$);
- 11: **Fin si**
- 12: $V2 \leftarrow \varepsilon_{j-1}; T2 \leftarrow T + d_{j-1} + d_j^* + p;$
- 13: **Si** $(V^{Veh} + \varepsilon_j^* \leq C^{Veh}) \wedge (T2 \leq TMax)$ **alors**
- 14: $W2 \leftarrow \alpha \times (d_{j-1} + d_j^* + p) + \beta \times (\varepsilon_{j-1} + \varepsilon_j^*) + W^{Veh};$
- 15: $x2 \leftarrow 1; Next2 \leftarrow q;$
- 16: **Veh_Bellman_Update**($STATE, NState, j, s2, W2, x2, Next2$);
- 17: **Fin si**
- 18: **fin pour**
- 19: **fin pour**

- $TInf_q$ est la date au plus tôt où la q^{ieme} opération de recharge peut commencer.
- $TSup_q$ est la date la plus tardive possible (i.e. la date au plus tard) à laquelle la q^{ieme} opération de recharge peut commencer;
- Δ_q est le délai minimal entre la date à laquelle la q^{ieme} opération de recharge en hydrogène peut commencer et la date à laquelle l'opération de recharge suivante ($q + 1$) peut commencer (ou la fin du voyage si $q = Q$).

3. Pour $q = 0$, on a $TInf_0 = 0$, $\Delta_0 = TInf_1$ et $TSup_0 = TMax - T$, où T est la valeur temps associée à l'état initial optimal.

L'algorithme (10) décrit le procédé de calcul de la stratégie de recharge primaire. Afin de synchroniser cette stratégie de recharge avec le processus de production d'hydrogène, nous devons transformer (à l'aide d'un processus de routine) les valeurs $TInf$, $TSup$ et Δ en termes de périodes $i = 0, \dots, N - 1$. C'est-à-dire de telle sorte que les vecteurs $TInf$, $TSup$ et Δ devront représenter le temps en nombre de périodes et non en unités de temps comme c'est le cas actuellement. Si par exemple $\Delta = 8$ et que la duré d'un période $p = 2$ alors Δ devra valoir $[8/2] = 4$ périodes au lieu de 8 unités de temps.

Exemple 4 Reprenons l'exemple (2), la **stratégie de recharge primaire** de cette instance est présenté au tableau (6.7).

Algorithme 10 Stratégie-de-recharge-primaire**Entrées:** $STATE, NState, E_0, TMax$ **Sorties:** $Q, LOAD$ **Initialisation :**

1: $Q \leftarrow 0; TCurr \leftarrow 0;$
 2: $q \leftarrow q_0; T_0 \leftarrow STATE_{0,q_0}.s.T; V_0 \leftarrow STATE_{0,q_0}.s.V^{Veh};$

Boucle principale :

3: **pour** $j = 0$ à M **faire**
 4: Soit $STATE_{j,q} = ((T, V^{Veh}), W^{Veh}, x, Next)$ et $STATE_{j+1,Next} = ((T_1, V1^{Veh}), W1, x1, Next1)$
 5: **Si** $x = 1$ **alors**
 6: **Si** $Q \geq 1$ **alors**
 7: $Taux \leftarrow LOAD_Q.TInf; Vaux \leftarrow 0;$
 8: **sinon**
 9: $Taux \leftarrow 0; Vaux \leftarrow E_0 - V_0;$
 10: **Fin si**
 11: $Q \leftarrow Q + 1;$
 12: $LOAD_Q \leftarrow (j, (V1^{Veh} + \varepsilon^* j + 1 - Vaux), (T_0 - T + d_j), (TMax - T1 - p - d_{j+1}^*), Nil); /*$
 Nil signifie ici indéfini */
 13: **Fin si**
 14: $q \leftarrow Next;$
 15: **fin pour**

q	St_q	L_q	$TInf_q$	$TSup_q$	Δ_q
1	3	12	19	41	14
2	4	22	33	55	*

TABLE 6.7 – La stratégie de recharge primaire de l'instance du modèle VD du tableau (6.2).

Exemple 5 Reprenons l'exemple (3), la **stratégie de recharge primaire** de cette instance est présenté au tableau (6.8).

q	St_q	L_q	$TInf_q$	$TSup_q$	Δ_q
1	0	26	1	47	27
2	2	8	28	74	11
3	3	34	39	85	37
4	7	28	76	122	*

TABLE 6.8 – La stratégie de recharge primaire de l'instance du modèle VD du tableau (6.4).

Après avoir extrait une stratégie de recharge primaire, nous devons la transformer en un format qu'on placera en entrée de l'algorithme **DPS_PM** présenté à la section suivante. On appelle cette seconde stratégie la **Stratégie de recharge réduite**. Pour cela, nous allons maintenant transformer les valeurs $TInf$, $TSup$ et Δ pour qu'elles représentent des dates en terme de périodes $i = 0, \dots, N - 1$. Cela signifie que nous allons dériver du vecteur $LOAD$:

- Q est le nombre d'opérations de recharge en carburant effectuées par le véhicule ;
- μ_q est la quantité d'hydrogène qui est rechargée lors de la q^{ieme} opération de recharge ;
- Des bornes inférieures m_1, \dots, m_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en hydrogène ont lieu ;

- Des bornes supérieures M_1, \dots, M_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N-1\}$ lorsque les opérations de recharge en hydrogène ont lieu ;
- Des coefficients de décalage B_1, \dots, B_{Q-1} qui reflètent les contraintes que ces numéros de période doivent satisfaire : pour tout $q = 1, \dots, Q-1$, $i_{q+1} \geq i_q + B_q$.

Nous désignons par μ_q la quantité d'hydrogène qui est chargée pour chaque valeur $q = 1, \dots, Q$. Ces vecteurs sont calculés par l'algorithme (11).

Algorithme 11 Stratégie-de-recharge-réduite

Entrées: $Q, L, TInf, TSup, \Delta, p, N, TMax$

Sorties: μ, m, M, B

Initialisation :

- 1: $m_1 \leftarrow \lceil TInf_1/p \rceil$;
- 2: $M_Q \leftarrow N - \lceil (TMax - TSup_Q)/p \rceil$;

Boucle principale :

- 3: **pour** $q = 1$ à Q faire
 - 4: $B_q \leftarrow \lceil \Delta_q/p \rceil$;
 - 5: $\mu_q \leftarrow L_q$;
 - 6: **fin pour**
 - 7: **pour** $q = 1$ à $Q-1$ faire
 - 8: $m_{q+1} \leftarrow m_q + B_q$
 - 9: **fin pour**
 - 10: **pour** $q = Q-1$ à 1 faire
 - 11: $M_q \leftarrow M_{q+1} - B_q$
 - 12: **fin pour**
-

Exemple 6 Reprenons l'exemple (2), la **stratégie de recharge réduite** de cette instance est présenté au tableau (6.9).

q	μ_q	m_q	M_q	B_q
1	12	5	9	4
2	22	9	13	*

TABLE 6.9 – La stratégie de recharge réduite de l'instance du modèle VD du tableau (6.2).

Exemple 7 Reprenons l'exemple (3), la **stratégie de recharge réduite** de cette instance est présenté au tableau (6.10).

q	μ_q	m_q	M_q	B_q
1	26	1	7	6
2	8	7	13	3
3	34	10	16	8
4	28	18	24	*

TABLE 6.10 – La stratégie de recharge réduite de l'instance du modèle VD du tableau (6.4).

6.4.6 Calcul des bornes inférieures du temps et de l'énergie du DPS_SMEPC

Notez qu'en exécutant l'algorithme (9) dans le cas $\alpha = 0$ et $\beta = 1$, cela nous permet de stocker, pour tout j et toute charge courante V^{Veh} , la quantité minimale d'énergie nécessaire pour finir

la tournée du véhicule. Et, d'utiliser cette information comme un outil de filtrage logique pour le traitement de l'algorithme (2) **DPS_SMEPC** du chapitre précédent.

De plus, en exécutant l'algorithme (9) dans le cas $\alpha = 1$ et $\beta = 0$, cela nous permet de stocker, pour tout j et toute charge courante V^{Veh} , **la quantité minimale de temps nécessaire pour finir la tournée du véhicule.** Et, d'utiliser cette information comme un outil de filtrage logique pour le traitement de l'algorithme (2) **DPS_SMEPC** du chapitre précédent.

Exécutons l'algorithme (9) avec $\alpha = 1$ et $\beta = 0$, et désignons par $STATE^T$ le vecteur résultant. De la même manière, exécutons l'algorithme (9) avec $\alpha = 0$ et $\beta = 1$, et désignons par $STATE^E$ le vecteur résultant. Ensuite, pour tout j , et pour toutes valeurs d'énergie V_0^{Veh} susceptibles de se produire dans le **DPS_SMEPC** :

- Désignons par W_T la plus petite valeur $STATE_{j,q}^T \cdot W^{Veh}$, $q = 1, \dots, NState_j$ obtenu avec q tel que : $STATE_{j,q} \cdot s = (j, V^{Veh})$ avec $V^{Veh} \geq V_0^{Veh}$; alors W_T est une borne inférieure pour le temps nécessaire pour aller jusqu'au dépôt final en partant de j avec un réservoir chargé de V_0^{Veh} ;
- Désignons par W_E la plus petite valeur $STATE_{j,q}^E \cdot W^{Veh}$, $q = 1, \dots, NState_j$ obtenue avec q telle que : $STATE_{j,q} \cdot s = (j, V^{Veh})$ avec $V^{Veh} \geq V_0^{Veh}$; alors W_E est une borne inférieure pour l'énergie que le véhicule doit utiliser pour aller jusqu'au dépôt final en partant de j avec V_0^{Veh} .

6.5 L'algorithme de programmation dynamique DPS_PM

Nous allons décrire ici l'algorithme de programmation dynamique **DPS_PM** que nous proposons pour résoudre le problème **PM**.

Nous supposons que les stations j telles que le véhicule se recharge entre la station j et station $j+1$ sont connues. Q indique le nombre de ces opérations de recharge. Nous supposons que les quantités d'hydrogène rechargée correspondantes μ_q , $q = 1, \dots, Q$ sont connues. Ainsi que les bornes inférieures m_1, \dots, m_Q et les bornes supérieures M_1, \dots, M_Q pour les numéros de période i_1, \dots, i_Q lorsque ces opérations de recharge ont lieu. Ainsi que les coefficients de décalage B_1, \dots, B_Q de telle sorte que nous puissions imposer les contraintes suivantes à ces numéros de période : pour tout $q = 1, \dots, Q-1$, $i_{q+1} \geq i_q + B_q$. Enfin, nous supposons également que nous disposons d'un coefficient λ .

La sortie doit être constituée de 2 vecteurs z de valeur $\{0, 1\}$ et δ avec une indexation sur $i = 0, \dots, N-1$ et la signification suivante :

- $z_i = 1$ signifie que la micro-usine produit pendant la période i ;
- $\delta_i = 1$ signifie qu'une opération de recharge en hydrogène a lieu pendant la période i ;

Notez que les variables d'activation de la micro-usine y_i prennent leurs valeurs selon la formule suivante : Si $i = 0$, alors $y_i = z_i$ sinon $y_i = z_i \times (1 - z_{i-1})$.

Rappelons que le problème **PM** est NP-Hard.

6.5.1 Espace temps et états au sens de la programmation dynamique

L'espace temps au sens de la programmation dynamique est l'ensemble $I = \{0, \dots, N\}$, qui va être parcouru de 0 à N .

Pour tout $i = 0, \dots, N$, un état est défini comme étant un quadruplet $E = (Z, V^{Tank}, Rank, Gap)$, avec $Rank \in \{1, \dots, Q\}$. E est défini de la façon suivante :

- $Z = 1$ signifie que la micro-usine produit au début de la période i (à la fin de la période $i - 1$) ;
- V^{Tank} est la quantité d'hydrogène qui se trouve dans le réservoir de la micro-usine au début de la période i ;
- $Rank \in \{1, \dots, Q\}$ signifie que l'opération de recharge qui a le numéro $Rank$ a été effectuée et que nous attendons pour effectuer la prochaine opération de recharge dont le numéro est $Rank + 1$;
- La valeur Gap est la différence entre i et la période durant laquelle la $Rank^{ieme}$ opération de recharge a été effectuée.

Pour chaque $i = 0, \dots, N$, un état E est fourni avec sa valeur de Bellman actuelle W^{Prod} .

Etat initial et Etats finaux

L'état initial est $E^{Start} = (0, H_0, 0, 0)$, avec la valeur $W^{Prod} = 0$, et la valeur $i = 0$; L'état final est l'état $E^{End} = (Z, V^{Tank} \geq H_0, Q, 0)$, associé à une valeur de temps $i \leq N$: veuillez noter que le processus peut ne pas être terminé lorsque la dernière opération de recharge a lieu, car la micro-usine peut devoir continuer à produire pour atteindre son niveau de charge initial H_0 .

6.5.2 Variables de décision

Pour tout $i = 0, \dots, N - 1$, $E = (Z, V^{Tank}, Rank, Gap)$, une décision est définie comme un couple (z, δ) dans $\{0, 1\}^2$, avec la signification suivante :

- $z = 1$ signifie que la micro-usine produira pendant la période i ;
- $\delta = 1$ signifie que le véhicule effectuera sa $(Rank + 1)^{ieme}$ opération de recharge en hydrogène pendant la période i .

6.5.3 Pré-conditions, transitions et coûts liés aux décisions et aux transitions

Comme la production et la recharge ne peuvent pas être effectuées simultanément, nous constatons qu'il n'y a que 3 décisions possibles. Ces décisions sont soumises aux conditions préalables suivantes :

- $z = 0, \delta = 0$: nous devrions avoir $(i \leq M_{Rank+1} - 1)$. Ce qui veut dire que la micro-usine et le véhicule ont décidé de ne rien faire (ni produire, ni recharger) à la période i , on doit avoir suffisamment de temps pour satisfaire la prochaine opération de recharge dont le numéro est $(Rank + 1)$;
- $z = 1, \delta = 0$: nous devrions avoir $(i \leq M_{Rank+1} - 1) \wedge (V^{Tank} + R_i \leq C^{Tank})$. Ce qui veut dire que la micro-usine produit à la période i et le véhicule ne se recharge pas à la période i , on doit avoir suffisamment de temps pour satisfaire la prochaine opération de recharge dont le numéro est $(Rank + 1)$. Et, la citerne doit avoir suffisamment d'espace libre pour stocker R_i quantités d'hydrogène : $(V^{Tank} + R_i \leq C^{Tank})$;
- $z = 0, \delta = 1$: nous devrions avoir $(V^{Tank} \geq \mu_{Rank+1}) \wedge (M_{Rank+1} > i \geq m_{Rank+1}) \wedge (Gap \geq B_{Rank})$. Ce qui veut dire que le véhicule se recharge à la période i , on doit avoir suffisamment d'hydrogène dans la citerne pour satisfaire la $(Rank + 1)^{ieme}$ opération de recharge : $(V^{Tank} \geq \mu_{Rank+1})$. Et, la recharge doit commencer à une date comprise entre m_{Rank+1} et $M_{Rank+1} - 1$ inclus : $(M_{Rank+1} > i \geq m_{Rank+1})$. Et, le décalage temporel entre la $Rank^{ieme}$ et la $(Rank + 1)^{ieme}$ opération de recharge doit être suffisamment long pour permettre la recharge : $(Gap \geq B_{Rank})$.

$1)^{i\text{eme}}$ opération de décalage doit satisfaire les coefficients de décalage correspondant données en entrées : ($\text{Gap} \geq B_{\text{Rank}}$).

Si ces décisions sont réalisables, elles induisent les transitions et les coûts suivants :

1. $z = 0, \delta = 0$

A la date $(i + 1)$, l'état résultant $E1$ sera :

- Si $\text{Rank} \leq Q - 1$ alors $E1 = (0, V^{\text{Tank}}, \text{Rank}, \text{Gap} + 1)$, et le coût de transition associé est égal à λ ;
- Si $\text{Rank} = Q$, alors $E1 = (0, V^{\text{Tank}}, \text{Rank}, 0)$, et le coût de transition correspondant est égal à 0 ;

2. $z = 1, \delta = 0$

A la date $(i + 1)$, l'état résultant $E1$ sera :

- Si $\text{Rank} \leq Q - 1$ alors $E1 = (1, V^{\text{Tank}} + R_i, \text{Rank}, \text{Gap} + 1)$, et le coût de transition associé est égal à $\lambda + (\text{Cost}^F \times (1 - Z) + \text{Cost}_i^V)$;
- Si $\text{Rank} = Q$, alors $E1 = (1, V^{\text{Tank}} + R_i, \text{Rank}, 0)$, et le coût de transition correspondant est égal à $(\text{Cost}^F \times (1 - Z) + \text{Cost}_i^V)$;

3. $z = 0, \delta = 1$

A la date $(i + 1)$, l'état résultant $E1$ sera $E1 = (0, V^{\text{Tank}} - \mu_{\text{Rank}+1}, \text{Rank} + 1, 1)$, et le coût de transition correspondant est égal à λ si $\text{Rank} + 1 \leq Q - 1$ sinon le coût de transition vaut zéro.

Exemple 8 Considérons la micro-usine de la figure (4.6), ainsi que l'entrée de la figure (4.7) : $p = 2$, $H_0 = 4$, $TMax = 30$, $\text{Cost}^F = 7$, $C^{\text{Tank}} = 15$, $\lambda = 2$, ainsi que la stratégie de recharge réduite du tableau (6.11) .

q	μ_q	m_q	M_q	B_q
0	*	0	3	3
1	14	3	6	8
2	11	11	14	*

TABLE 6.11 – Exemple de stratégie de recharge réduite.

Supposons que $i = 12$ et que l'état correspondant soit : $Z = 0$, $V^{\text{Tank}} = 12$, $\text{Rank} = 1$, $\text{Gap} = 9$. Nous voyons alors que les décisions suivantes sont possibles :

- $z = 0, \delta = 0$: L'état résultant est $Z = 0$, $V^{\text{Tank}} = 12$, $\text{Rank} = 1$, $\text{Gap} = 10$, avec un coût de transition de 2 ;
- $z = 1, \delta = 0$: l'état résultant est $Z = 1$, $V^{\text{Tank}} = 13$, $\text{Rank} = 1$, $\text{Gap} = 10$, avec un coût de transition de $7 + 2 + 2 = 11$;
- $z = 0, \delta = 1$: L'état résultant est $Z = 0$, $V^{\text{Tank}} = 1$, $\text{Rank} = 2$, $\text{Gap} = 1$, avec un coût de transition de 2.

6.5.4 Mise en œuvre algorithmique du DPS_PM

Comme structure de données nous utilisons :

1. Un vecteur à deux dimensions désigné par $PROD$, avec une indexation sur $i = 0, \dots, N$, et $q = 1, \dots, \text{Prod_State_Max}$, où Prod_State_Max est le nombre maximal d'état autorisées pour chaque valeur de temps i . Pour chaque $i \in I$ on a un ensemble $PROD_i$ d'états associés. ;

2. Un vecteur N_PROD , avec indexation sur $i = 0, \dots, N : N_PROD_i$ nous fournit le nombre d'états dans $PROD_i$.

Chaque composante d'état $PROD_{i,q}$ est un quadruplet $(E = (Z, V^{Tank}, Rank, Gap), W^{Prod}, (z, \delta), Pred)$ où :

- Chaque état $E = (Z, V^{Tank}, Rank, Gap)$ est donné avec une valeur de Bellman $W^{Prod} = \lambda \times \Pi + Cost$, où $Cost$ est le coût économique de production actuel du processus, et Π est le nombre de périodes déjà parcouru, tant que le véhicule n'a pas encore effectué toutes ses opérations de recharge
- (z, δ) est la décision prise au début de la période i ;
- $Pred$ est l'indice q^* tel que l'état actuel E découle de $PROD_{i-1,q^*}$ (on dit que $PROD_{i-1,q^*}$ est le père de E).

De plus, comme un état final peut être lié à des temps (au sens de la programmation dynamique) i différents de N , nous conservons un état final actuel $FINAL$, qui correspond à la meilleure situation (meilleure valeur) lorsque $Rank = Q$ et $V^{Tank} \geq H_0$. $FINAL$ est du même type que les composantes $PROD_{i,k}$ du vecteur $PROD$, augmentées de la valeur i :

- $FINAL = ((E = (Z, V^{Tank}, Q, 0), W^{Prod}, (z, \delta), Pred))$;
- $FINAL_TIME$ est la valeur i telle que le processus est terminé au début de la période i .

Equations de Bellman

Lorsqu'on veut ajouter un nouvel état E^* dont la valeur est W^* et le prédecesseur q dans le vecteur $PROD_i$ on a deux possibilités :

1. L'état E^* n'existe pas dans $PROD_i$ auquel cas on l'ajoute à $PROD_i$ en incrémentant le nombre d'états N_PROD_i de la liste.
2. L'état E^* existe déjà dans $PROD_i$ dans ce cas on applique la procédure de Bellman pour garder le meilleure état. Dans notre cas, on garde l'état qui a la plus petite valeur.

L'algorithme (12) décrit la procédure de Bellman utilisée pour ajouter un nouvel état E^* dans le vecteur $PROD$.

Stratégie d'exploration

La stratégie de parcours du temps au sens de la programmation dynamique utilisé ici est la stratégie en *Forward*.

Algorithme DPS_PM

Les entrées de l'algorithme **DPS_PM** sont $C^{Tank}, N, H_0, R, Cost^F, Cost^V$ et λ et, les sorties sont $PROD$ et $FINAL$. Le schéma de programmation dynamique **DPS_PM** qui fonctionne en *Forward* est développé par l'algorithme (13).

6.5.5 Mécanismes de filtrage du DPS_PM

Pour diminuer le nombre d'états générés par le DPS décrit à l'algorithme (13), on a implémenté trois types de mécanismes de filtrage : des mécanismes de filtrage par arrondi, des mécanismes de filtrage par relations de dominance et des mécanismes de filtrage logique.

Algorithme 12 Prod_Bellman_Update

Entrées: PROD, FINAL, i, q, DEC, E*, W***Sorties:** PROD, FINAL,**Initialisation :**

1: NotStop;

Boucle principale :2: **Si** ($E^*.V^{Tank} \geq H_0$) \wedge ($E^*.Rank \geq Q$) **alors**3: **Si** $W^* \leq FINAL.W^{Prod}$ **alors**4: $FINAL \leftarrow (E^*, W^*, DEC, q)$; /* E^* est un état final possible */5: $FINAL_TIME \leftarrow i + 1$;6: **Fin si**7: **sinon**8: $q1 \leftarrow 1$;9: **Tant que** ($q1 \leq N_PROD_{i+1}$) \wedge NotStop **faire**10: **Si** $PROD_{i+1,q1}.E = E^*$ **alors**11: $Stop$;12: **sinon**13: $q1 \leftarrow q1 + 1$;14: **Fin si**15: **Fin tant que**16: **Si** NotStop **alors**17: $N_PROD_{i+1} \leftarrow N_PROD_{i+1} + 1$;18: $PROD_{i+1,N_PROD_{i+1}} \leftarrow (E^*, W^*, DEC, q)$; /* E^* n'est pas présent dans $PROD_{i+1,q1}$ */19: **sinon**20: $WAux \leftarrow PROD_{i+1,q1}.W^{Prod}$;21: **Si** $W^* < WAux$ **alors**22: $PROD_{i+1,q1} \leftarrow (E^*, W^*, DEC, q)$; /* E^* est déjà présent dans $PROD_{i+1,q1}$ */23: **Fin si**24: **Fin si**25: **Fin si**

Algorithme 13 DPS_PM**Entrées:** C^{Tank} , N , H_0 , R , $Cost^F$, $Cost^V$, λ **Sorties:** $PROD$, $FINAL$ **Initialisation :**

```

1:  $i \leftarrow 0$ ;  $N\_PROD_0 \leftarrow 1$ ;  $PROD_{0,0} \leftarrow ((0, H_0, 0, 0), 0, Nil, Nil)$ ; /* Nil signifie ici indéfini */
2:  $FINAL \leftarrow (Nil, +\infty, Nil, Nil)$ ;  $FINAL\_TIME \leftarrow Nil$ ;
3: pour  $i = 1$  à  $N$  faire
4:    $N\_PROD_i \leftarrow 0$ ;
5: fin pour

```

Boucle principale :

```

6: pour  $i = 0$  à  $N - 1$  faire
7:   pour  $q = 0$  à  $N\_PROD_i - 1$  faire
8:     Soit  $PROD_{i,q} = (E = (Z, V^{Tank}, Rank, Gap), W^{Prod}, (z, \delta), Pred)$ ;
9:      $DEC1 \leftarrow (0, 0)$ ; /* 1ière décisions : La micro-usine ne produit pas à la période  $i$  */
10:    Si  $Rank = Q$  alors
11:       $E1 \leftarrow (0, V^{Tank}, Rank, 0)$ ;
12:    sinon
13:       $E1 \leftarrow (0, V^{Tank}, Rank, Gap + 1)$ ;
14:    Fin si
15:    Si  $Rank \leq Q - 1$  alors
16:       $W1 \leftarrow W^{Prod} + \lambda$ ;
17:    sinon
18:       $W1 \leftarrow W^{Prod}$ ;
19:    Fin si
20:    Si  $(i + 1) \leq M_{Rank+1}$  alors
21:      Prod_Bellman_Update( $PROD$ ,  $FINAL$ ,  $i$ ,  $q$ ,  $DEC1$ ,  $E1$ ,  $W1$ );
22:    Fin si
23:     $DEC2 \leftarrow (1, 0)$ ; /* 2ième décisions : La micro-usine produit à la période  $i$  */
24:    Si  $Rank = Q$  alors
25:       $E2 \leftarrow (1, V^{Tank} + R_i, Rank, 0)$ ;
26:    sinon
27:       $E2 \leftarrow (1, V^{Tank} + R_i, Rank, Gap + 1)$ ;
28:    Fin si
29:    Si  $Rank \leq Q - 1$  alors
30:       $W2 \leftarrow W^{Prod} + (Cost^F \times (1 - Z) + Cost_i^V) + \lambda$ ;
31:    sinon
32:       $W2 \leftarrow W^{Prod} + (Cost^F \times (1 - Z) + Cost_i^V)$ ;
33:    Fin si
34:    Si  $(V^{Tank} + R_i \leq C^{Tank}) \wedge ((i + 1) \leq M_{Rank+1})$  alors
35:      Prod_Bellman_Update( $PROD$ ,  $FINAL$ ,  $i$ ,  $q$ ,  $DEC2$ ,  $E2$ ,  $W2$ );
36:    Fin si
37:     $DEC3 \leftarrow (0, 1)$ ; /* 3ième décisions : Le véhicule se recharge à la période  $i$  */
38:     $E3 \leftarrow (0, V^{Tank} - \mu_{Rank+1}, Rank + 1, 1)$ ;
39:    Si  $Rank + 1 \leq Q - 1$  alors
40:       $W3 \leftarrow W^{Prod} + \lambda$ ;
41:    sinon
42:       $W3 \leftarrow W^{Prod}$ ;
43:    Fin si
44:    Si  $(V^{Tank} - \mu_{Rank+1} \geq 0) \wedge (M_{Rank+1} \geq i) \wedge (i \geq m_{Rank+1}) \wedge (Gap \geq B_{Rank})$  alors
45:      Prod_Bellman_Update( $PROD$ ,  $FINAL$ ,  $i$ ,  $q$ ,  $DEC3$ ,  $E3$ ,  $W3$ );
46:    Fin si
47:  fin pour
48: fin pour

```

Mécanismes de filtrage par arrondi

Comme à la section 5.3, nous pouvons arrondir les valeurs V^{Tank} et W^{Prod} et transformer DPS_PM en un algorithme paramétré DPS_PM(K) qui est polynomial dans le temps pour toute valeur K fixe, et tel que, pour toute valeur $\varepsilon > 0$, K peut être choisi de telle sorte que dans le cas où Tank admet une solution optimale avec la valeur W^{Prod_Opt} , alors DPS_PM(K) donne une solution qui est réalisable par rapport à la valeur initiale $(1 + \varepsilon/2) \times H_0$ et les valeurs de capacité $(1 + \varepsilon) \times C^{Tank}$, et dont la valeur de coût n'est pas supérieure à W^{Prod_Opt} .

Mécanismes de filtrage par relations de dominance

Pour une valeur de temps (au sens de la programmation dynamique) donnée i , si deux états $E_1 = (Z_1, V_1^{Tank}, Rank_1, Gap_1)$ et $E_2 = (Z_2, V_2^{Tank}, Rank_2, Gap_2)$ donnés avec les valeurs W_1^{Prod} et W_2^{Prod} sont tels que :

- $W_1^{Prod} \leq W_2^{Prod}$;
- $V_1^{Tank} \geq V_2^{Tank}$;
- $Rank_1 \geq Rank_2$; Si $Rank_1 = Rank_2$, alors $Gap_1 \geq Gap_2$
- $Z_1 \geq Z_2$;

on peut alors dire que E_1 domine E_2 , et donc que E_2 peut être retiré de la liste des états associés à i .

Mécanismes de filtrage logique

On liste les mécanismes de filtrage qui concerne la faisabilité du processus au regard des contraintes énergétiques. Etant donné une valeur de temps (au sens de la programmation dynamique) i avec un état $E = (Z, V^{Tank}, Rank, Gap)$. Désignons par $Prod_Max(i) = \sum_{k \geq i} R_k$, l'énergie maximale que la micro-usine peut produire tout en commençant à produire à la période i . Nous constatons que :

- Si $V^{Tank} + \sum_{M_Q > k \geq i} R_k < \sum_{Rank+1 \leq q \leq Q} \mu_q$, alors nous pouvons tuer l'état E lié au temps i car il est impossible de produire suffisamment d'hydrogène pour satisfaire toutes les q^{ieme} recharges, avec $Rank + 1 \leq q \leq Q$.
- De la même manière, si $V^{Tank} + Prod_Max(i) < \sum_{Rank+1 \leq q \leq Q} \mu_q + H_0$, nous pouvons également tuer l'état E lié au temps i .

En pratique, le vecteur $Prod_Max$ doit être calculé dans le cadre d'un pré-processus, de manière à obtenir un accès direct aux quantités $Prod_Max(i)$.

Mécanismes de filtrage par estimation optimiste

Elle implique, comme à la section 5.4.3, la fonction pré-calculée $CostMin(i, V^{Tank}, Z)$, qui signifie pour un temps (au sens de la programmation dynamique) i et toute paire (Z, V^{Tank}) , la valeur $CostMin(i, V^{Tank}, Z)$ est le coût minimum requis pour que la micro-usine produise V^{Tank} unités d'énergie pendant les périodes $i, i+1, \dots, N-1$ (i.e entre le temps $p.i$ et le temps $TMax$), Z désignant l'état de la micro-usine au début de la période i . Supposons maintenant que nous disposions d'une solution actuelle réalisable de notre problème PM, avec la valeur W_{Curr}^{Prod} et que nous ayons affaire, à la date i , à un état $E = (Z, V^{Tank}, Rank, Gap)$ et une valeur W^{Prod} connexe. Si :

$W^{Prod} + CostMin(i, \sum_{q \geq Rank+1} \mu_q + H_0 - V^{Tank}, Z) + \lambda \times (m_Q + i - (Gap + m_{Rank})) \geq W_{Curr}^{Prod}$, nous pouvons tuer l'état $E = (Z, V^{Tank}, Rank, Gap)$ lié au temps i .

Remarque 4 $i - Gap - m_{Rank}$ représente le décalage entre la période de la dernière recharge effectuée et sa période minimale.

Calcul de la borne supérieure W_{Curr}^{Prod} pour le problème PM

$CostMin$ nous permet de transformer le **DPS_PM** en un algorithme glouton **Greedy_PM** (Voir algorithme (14)) :

- Pour tout i et tout état $E = (Z, V^{Tank}, Rank, Gap)$ lié à i , la quantité d'hydrogène qui reste à produire est $V = \sum_{q \geq Rank+1} \mu_q + H_0 - V^{Tank}$;
- De même, la Q^{ieme} opération de recharge en carburant ne peut avoir lieu avant la période $m_Q + i - Gap - m_{Rank}$;
- Ainsi, **Greedy_PM** fonctionne en ne conservant, pour tout i , qu'un seul état E , et en choisissant une décision réalisable connexe (z, δ) de telle sorte que l'état résultant $E_1 = (Z_1, V_1^{Tank}, Rank_1, Gap_1)$ soit cohérent avec la règle de filtrage logique ci-dessus et que le $CostMin(i+1, \sum_{q \geq Rank_1+1} \mu_q + H_0 - V_1^{Tank}, Z_1) + \alpha \times (m_Q - Gap_1 - m_{Rank_1}) + (\text{Coût de la transition } (i, E) \rightarrow (i+1, E_1))$ soit minimal.

L'algorithme glouton (14) peut être appliqué afin de nous fournir (en cas de succès) une stratégie de production initiale (z, δ) et sa valeur W_{Curr}^{Prod} . Le calcul de W_{Curr}^{Prod} est fait en appliquant l'algorithme (13) de telle sorte que, pour tout i , l'ensemble $PROD_i$ ne contienne qu'un seul état, choisi de manière heuristique.

On va considérer qu'on a trois variables de filtrage VAL_TESTa , VAL_TEST_a et VAL_TEST_a , avec $a \in \{1, 2, 3\}$. Elles vont nous servir à vérifier que chaque état respecte les mécanismes de filtrage. La signification des variables de filtrage est :

- VAL_TESTa est l'estimation du coût (temps et coût de production) anticipé pour satisfaire toutes les recharges ;
- VAL_TEST_a est la quantité d'hydrogène maximale qu'on peut produire avant la date $TMax$ en commençant la production à la période actuelle ;
- VAL_TEST_a est la quantité d'hydrogène maximale qu'on peut produire avant la dernière recharge en commençant la production à la période actuelle.

En fonction de la décision prise les valeurs VAL_TESTa , VAL_TEST_a , VAL_TEST_a changent :

Si on prend la décision $DEC = (0, 0)$ on a :

- $VAL_TEST1 = CostMin(i+1, \sum_{q \geq Rank+1} \mu_q + H_0 - V^{Tank}, 0) + \lambda \times (m_Q + i + 1 - Gap - m_{Rank})$
- $VAL_TEST_1 = (V^{Tank} + Prod_Max(i+1) - \sum_{Rank+1 \leq q \leq Q} \mu_q + H_0)$;
- $VAL_TEST_1 = V^{Tank} + Prod_Max(i+1) - Prod_Max(M_Q) - \sum_{Rank+1 \leq q \leq Q} \mu_q$

Ceci correspond à l'instruction 21 de l'algorithme (15).

Si on prend la décision $DEC = (1, 0)$ on a :

- $VAL_TEST2 = CostMin(i+1, \sum_{q \geq Rank+1} \mu_q + H_0 - V^{Tank} - R_i, 1) + \lambda \times (m_Q + i + 1 - Gap - m_{Rank})$;

Dans ce cas, VAL_TEST_2 et VAL_TEST_2 ne sont pas calculées. Ceci correspond à l'instruction 36 de l'algorithme (15).

Si on prend la décision $DEC = (0, 1)$ on a :

Algorithme 14 Greedy_PM

Entrées: C^{Tank} , N , H_0 , R , $Cost^F$, $Cost^V$, λ **Sorties:** W^{Prod} , DEC_Prod **Initialisation :**

1: $i \leftarrow 0$; $E \leftarrow (0, H_0, 0, 0)$; $NotFailure$; $Notsuccess$;
 2: $W^{Prod} \leftarrow 0$; $DEC_Prod \leftarrow Nil$;

Boucle principale :

3: **Tant que** $NotFailure \wedge Notsuccess$ **faire**
 4: Soit $E = (Z, V^{Tank}, Rank, Gap)$;
 5: $DEC1 \leftarrow (0, 0)$; /* **La micro-usine ne produit pas à la période i** */
 6: **Exécuter** les instructions de 10 à 19 de l'algorithme (13);
 7: **Calculer** $VAL_TEST1, VAL_TEST_1, VAL_TEST_1$; /*Voir section 6.5.5*/
 8: **Si** $((i + 1) \leq M_{Rank+1}) \wedge (VAL_TEST_1 \geq 0) \wedge (VAL_TEST_1 \geq 0)$ **alors**
 9: $EVAL1 \leftarrow VAL_TEST1 + W1$;
 10: **sinon**
 11: $EVAL1 \leftarrow +\infty$;
 12: **Fin si**
 13: $DEC2 \leftarrow (1, 0)$; /* **La micro-usine produit à la période i** */
 14: **Exécuter** les **instructions** de 24 à 33 de l'algorithme (13);
 15: **Calculer** VAL_TEST2 ; /*Voir section 6.5.5*/
 16: **Si** $(V^{Tank} + R_i \leq C^{Tank}) \wedge ((i + 1) \leq M_{Rank+1})$ **alors**
 17: $EVAL2 \leftarrow VAL_TEST2 + W2$;
 18: **sinon**
 19: $EVAL2 \leftarrow +\infty$;
 20: **Fin si**
 21: $DEC3 \leftarrow (0, 1)$; /* **Le véhicule se recharge à la période i** */
 22: **Exécuter** les **instructions** de 38 à 43 de l'algorithme (13);
 23: **Calculer** $VAL_TEST3, VAL_TEST_3, VAL_TEST_3$; /*Voir section 6.5.5*/
 24: **Si** $(V^{Tank} - \mu_{Rank+1} \geq 0) \wedge (M_{Rank+1} \geq i \geq m_{Rank+1}) \wedge (Gap \geq B_{Rank}) \wedge (VAL_TEST_3 \geq 0) \wedge (VAL_TEST_3 \geq 0)$ **alors**
 25: $EVAL3 \leftarrow VAL_TEST3 + W3$;
 26: **sinon**
 27: $EVAL3 \leftarrow +\infty$;
 28: **Fin si**
 29: $EVAL \leftarrow Inf(EVAL1, EVAL2, EVAL3)$;
 30: **Si** $EVAL = +\infty$ **alors**
 31: $Failure$;
 32: **sinon**
 33: $DEC_Prod_i \leftarrow$ Décision $DEC1, DEC2, DEC3$ correspondant à $EVAL$;
 34: $W^{Prod} \leftarrow$ Valeur $W1, W2, W3$ correspondant à $EVAL$;
 35: $E \leftarrow E1, E2, E3$ correspondant à $EVAL$;
 36: **Si** $(Rank = Q) \wedge (E.V^{Tank} \geq H_0)$ **alors**
 37: $Success$;
 38: **Fin si**
 39: **Fin si**
 40: $i \leftarrow i + 1$;
 41: **Fin tant que**

- $VAL_TEST3 = CostMin(i + 1, \sum_{q \geq Rank+1} \mu_q + H_0 - V^{Tank}, 0) + \lambda \times (m_Q + i - m_{Rank+1})$
- $VAL_TEST_3 = (V^{Tank} + Prod_Max(i + 1) - \sum_{Rank+1 \leq Q \leq Q} \mu_Q + H_0);$
- $VAL_TEST_3 = V^{Tank} + Prod_Max(i + 1) - Prod_Max(M_Q) - \sum_{Rank+1 \leq q \leq Q} \mu_q$

Ceci correspond à l'instruction 47 de l'algorithme (15).

Nous réécrivons la procédure **DPS_PM** tout en y ajoutant les instructions supplémentaires qui font référence à l'application des mécanismes de filtrage basés sur la logique et par estimation optimiste (Voir algorithme (15)).

Mécanismes de filtrage heuristique

Notre objectif dans cette section est de décrire des mécanismes de filtrage heuristique qui serviront à diminuer le nombre d'états de l'algorithme **DPS_PM** quitte à induire un certain niveau d'approximation. On rappelle qu'un état d'un temps i est défini par un quadruplet $(Z, V^{Tank}, Rank, Gap)$. On fixe un paramètre entier K dit de fluidification (on pourra utiliser par exemple $K = 10$ ou $K = 5$). Lorsqu'on est à un temps i et que l'on vient de générer, à partir d'un état E_0 et d'une décision D un nouvel état $E = (Z, V^{Tank}, Rank, Gap)$ accompagné d'une valeur W et associé au temps i alors :

1. On applique tous les mécanismes de filtrage décrit précédemment ;
2. Si l'état E n'a pas été supprimé par ces mécanismes de filtrage alors on balaie l'ensemble des états du temps i et chaque fois que l'on rencontre un état $E_1 = (Z_1, V_1^{Tank}, Rank_1, Gap_1)$ accompagné d'une valeur W_1 , on procède comme suit :

Si $Z = Z_1$ et $|V^{Tank} - V_1^{Tank}| \leq (C^{Tank}/K)$ et $Rank = Rank_1$ alors on dit que E et E_1 sont équivalents modulo K :

- si $W < W_1$ alors on remplace E_1, W_1 par E, W dans les états du temps i ;
- dans le cas inverse, on ne fait rien, donc on n'insère pas E, W parmi les états du temps i ;

6.6 Collaboration Demandeur/Producteur pour SMEPC : le schéma Pipe-line VD_PM

Nous allons maintenant expliquer la manière dont le modèle **SMEPC** du chapitre précédent peut être décomposé de manière à reproduire la façon dont les décisions vont être prises dans le cas (réaliste) où la décision est prise en *collaboration*, ce qui signifie que le *directeur de production* et le *conducteur du véhicule* sont des acteurs indépendants, qui doivent interagir. L'idée principale ici est qu'un conducteur de véhicule décentralisé ait pour comportement naturel de se déplacer comme s'il était sûr d'avoir assez de carburant chaque fois qu'il se rend à la micro-usine, et de s'adapter au contexte réel en attendant et éventuellement en retardant certains déplacements. À l'inverse, un comportement naturel du directeur de production consistera à s'adapter à la demande, et à faire un compromis entre la qualité de service et le coût de production. Il est clair que notre schéma de décomposition Véhicule/Production, qui est un schéma de décomposition quelque peu hiérarchique avec le problème du Véhicule (problème **VD**) comme maître et le problème de la Production (problème **PM**) comme esclave, peut être transformé en un simple processus heuristique désigné **Pipe-line VD_PM**.

Algorithme 15 Filtered_DPS_PM**Entrées:** C^{Tank} , N , H_0 , R , $Cost^F$, $Cost^V$, λ **Sorties:** $PROD$, $FINAL$ **Initialisation :**

- 1: $i \leftarrow 0$; $N_PROD_0 \leftarrow 1$; $PROD_{0,0} \leftarrow ((0, H_0, 0, 0), 0, Nil, Nil)$;
- 2: $FINAL \leftarrow (Nil, +\infty, Nil, Nil)$; $FINAL_TIME \leftarrow Nil$; /* Nil signifie ici indéfini */
- 3: **pour** $i = 1$ à N **faire**
- 4: $N_PROD_i \leftarrow 0$;
- 5: **fin pour**
- 6: **Calculez** la borne supérieure W_{Curr}^{Prod} à l'aide la procédure 14 **Greedy_PM**;

Boucle principale :

- 7: **pour** $i = 0$ à $N - 1$ **faire**
- 8: **pour** $q = 0$ à $N_PROD_i - 1$ **faire**
- 9: Soit $PROD_{i,q} = (E = (Z, V^{Tank}, Rank, Gap), W^{Prod}, (z, \delta), Pred)$;
- 10: $DEC1 \leftarrow (0, 0)$; /* La micro-usine ne produit pas à la période i */
- 11: **Exécuter** les instructions de 10 à 19 de l'algorithme (13);
- 12: **Calculer** $VAL_TEST1, VAL_TEST_1, VAL_TEST_1$; /*Voir section 6.5.5*/
- 13: **Si** $((i + 1) \leq M_{Rank+1}) \wedge (VAL_TEST1 + W1 \leq W_{Curr}^{Prod}) \wedge (VAL_TEST_1 \geq 0) \wedge (VAL_TEST_1 \geq 0)$ **alors**
- 14: **Prod_Bellman_Update**($PROD, FINAL, i, q, DEC1, E1, W1$);
- 15: **Fin si**
- 16: $DEC2 \leftarrow (1, 0)$; /* La micro-usine produit à la période i */
- 17: **Exécuter** les **instructions** de 24 à 36 de l'algorithme (13);
- 18: **Calculer** VAL_TEST2 ; /*Voir section 6.5.5*/
- 19: **Si** $(V^{Tank} + R_i \leq C^{Tank}) \wedge ((i + 1) \leq M_{Rank+1}) \wedge (VAL_TEST2 + W2 \leq W_{Curr}^{Prod})$ **alors**
- 20: **Prod_Bellman_Update**($PROD, FINAL, i, q, DEC2, E2, W2$);;
- 21: **Fin si**
- 22: $DEC3 \leftarrow (0, 1)$; /* Le véhicule se recharge à la période i */
- 23: $E3 \leftarrow (0, V^{Tank} - \mu_{Rank+1}, Rank + 1, 1)$;
- 24: **Si** $Rank + 1 \leq Q - 1$ **alors**
- 25: $W3 \leftarrow W^{Prod} + \lambda$;
- 26: **sinon**
- 27: $W3 \leftarrow W^{Prod}$;
- 28: **Fin si**
- 29: **Calculer** $VAL_TEST3, VAL_TEST_3, VAL_TEST_3$; /*Voir section 6.5.5*/
- 30: **Si** $(V^{Tank} - \mu_{Rank+1} \geq 0) \wedge (M_{Rank+1} \geq i \geq m_{Rank+1}) \wedge (Gap \geq B_{Rank}) \wedge (VAL_TEST3 + W3 \leq W_{Curr}^{Prod}) \wedge (VAL_TEST_3 \geq 0) \wedge (VAL_TEST_3 \geq 0)$ **alors**
- 31: **Prod_Bellman_Update**($PROD, FINAL, i, q, DEC3, E3, W3$);;
- 32: **Fin si**
- 33: **fin pour**
- 34: **fin pour**

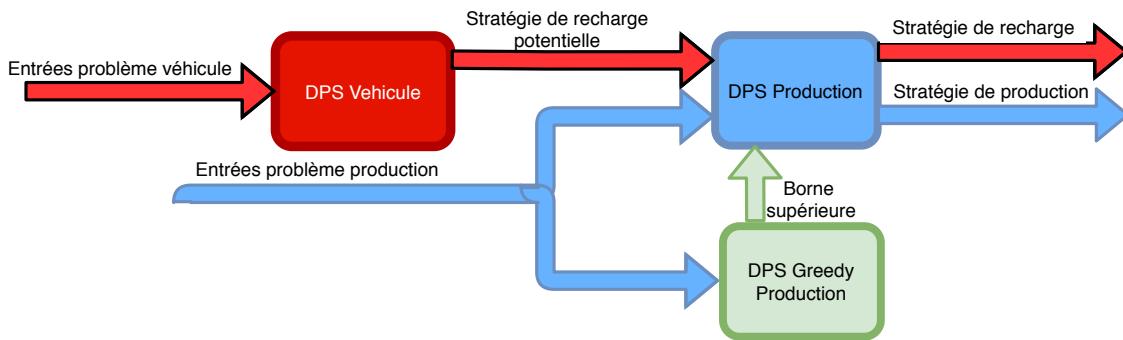


FIGURE 6.1 – Schéma du Pipe-line VD_PM.

La décomposition **Pipe-line VD_PM** (Voir figure (6.1)) suggère que la façon la plus simple de mettre en place une collaboration entre le conducteur du véhicule et le directeur de production est de les faire communiquer par le biais d'un pipeline à sens unique : une fois que le coefficient β ait été fixé. β détermine la fonction objectif du problème **VD** et se comporte comme le principal lien de communication entre le problème **VD** et le problème **PM**. La décomposition Pipe-line VD_PM fonctionne de la façon suivante : on calcule d'abord une stratégie de recharge optimale en carburant (x, L) pour le véhicule au moyen de l'algorithme **DPS_VD**, on transforme ensuite cette stratégie (x, L) en une stratégie de recharge réduite (Q, μ, m, M, B) et enfin on applique l'algorithme **DPS_PM** sur cette stratégie de recharge réduite.

Remarque 5 Nous faisons communiquer un DPS fonctionnant en Backward (**DPS_VD**) et un DPS fonctionnant en Forward (**DPS_PM**) car dans la littérature ces deux schémas communiquent « bien ».

Concernant les données d'entrées, nous supposons que nous recevons toutes les données relatives à la micro-usine et au véhicule, ainsi que le coefficient α . Concernant la sortie désigné ici par **solution complète**, elle est constituée par le vecteur de production $z = (z_i, i = 0, \dots, N - 1)$, le vecteur d'activation $y = (y_i, i = 0, \dots, N - 1)$, le vecteur de recharge du véhicule $\delta = (\delta_i, i = 0, \dots, N - 1)$, le vecteur de recharge du véhicule $x = (x_j, j = 0, \dots, M)$, le vecteur temps $T = (T_j, j = 0, \dots, M + 1)$, le vecteur de la quantité rechargée $L = (L_j, j = 0, \dots, M)$, qui nous indique, dans le cas où $x_j = 1$, quelle quantité d'hydrogène est rechargée et la valeur VAL .

Nous voyons ensuite que **SMEPC** peut être reformulée de manière collaborative comme suit : {Fixez les coefficients et calculez une **stratégie de recharge optimale** (x, L) de manière à ce que la *stratégie de recharge réduite* (Q, μ, m, M, B, μ) donne une stratégie de production (z, δ) avec un coût minimal de $\alpha \times p \times i_Q + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost^V \times z_i)$ }.

Les étapes de l'heuristique Pipe-line VD_PM (Voir algorithme 16) sont :

1. **Etape 1** : Fixer le coefficient β ;
2. **Etape 2** : Résoudre, par le biais de l'algorithme **DPS_VD**, l'instance du véhicule liée aux coefficients β et α et obtenir le nombre d'opérations de recharge Q ainsi que les vecteurs m , M et B ;
3. **Etape 3** : Calculer W_{Curr}^{Prod} ainsi qu'une solution initiale **DEC_PROD** de l'instance de production liée à $\lambda = \alpha \times p$, par le biais de l'algorithme **Greedy_PM** ;
4. **Etape 4** : Résolvez l'instance de production résultante liée à $\lambda = \alpha \times p$, par le biais de l'algorithme **Filtered_DPS_PM** ;

Algorithme 16 Pipe-line VD_PM**Entrées:** $N, M, TMax, H_0, E_0, C^{Tank}, C^{Veh}$,**Sorties:** $Current_Sol, Current_Value$

- 1: Fixer le coefficient β du modèle VD ;
- 2: Appliquez le **DPS_VD** au modèle VD et obtenez la **stratégie de recharge optimale** (x, L) et récupérez la **stratégie de recharge réduite** (Q, μ, m, M, B) comme dans la section IV.3 ;
- 3: Calculer W_{Curr}^{Prod} par le biais de l'algorithme **Greedy_PM** ;
- 4: Résolvez l'instance de production par le biais de l'algorithme **Filtered_DPS_PM** à l'aide de la **stratégie de recharge réduite** (Q, μ, m, M, B) et de la **stratégie de recharge optimale** (x, L) ;
- 5: **Reconstruire la solution complète** en récupérant le vecteur d'activation $y = (y_i, i = 0, \dots, N-1)$, le vecteur temps $T = (T_j, j = 0, \dots, M+1)$, le vecteur de charge $L = (L_i, i = 0, \dots, N-1)$, et le coût global $\alpha \times p \times i_Q + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i)$;

5. Etape 5 : Reconstruire la solution complète du problème SMEPC (à la fois la tournée du véhicule et l'activité de la micro-usine), ainsi que sa valeur VAL .

Le choix de λ vient de manière très naturelle : nous fixons $\lambda = \alpha \times p$. Nous devons d'abord préciser l'étape 5 : L'étape 2 nous a fourni une stratégie de recharge primaire, et donc les vecteurs L et x . Elle nous a également fourni le nombre Q d'opérations de recharge, et, à l'aide de l'état FINAL et des composantes *Pred* et *DEC*, nous dérivons de l'étape 4 les vecteurs z , y et δ . Afin d'obtenir le vecteur temps T , nous exécutons l'algorithme (17).

Algorithme 17 T_Reconstruction**Entrées:** M, t, d^*, p, δ **Sorties:** T **Initialisation :**1: $T_0 \leftarrow 0$; $i_aux \leftarrow 0$;**Boucle principale :**2: **pour** $j = 0$ à M **faire**3: **Si** $x_j = 0$ **alors**4: $T_{j+1} \leftarrow T_j + t_j$;5: **sinon**6: Calculer le plus petit $i_0 \geq i_aux$ tel que $\delta_{i_0} = 1$;7: $T_{j+1} \leftarrow p \times (i_0 + 1) + d_{j+1}^*$;8: $i_aux \leftarrow i_0 + 1$;9: **Fin si**10: **fin pour**

Ensuite, la valeur VAL est calculée d'une manière simple :

$$VAL = \alpha \times p \times [i_Q + d_{St_Q+1}^* + \sum_{j=St_Q+1, \dots, M-1} t_j + d_M] + \sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost_i^V \times z_i).$$

Nous devons discuter de la manière de choisir β car il doit refléter le coût de production de la quantité d'hydrogène $\sum_q \mu_q$ qui doit être produite pour permettre à la tournée augmentée $\Gamma(\beta)$ d'être effectuée. Cependant, nous n'avons aucune idée de ce que sera réellement ce coût, puisque nous ne connaissons pas a priori la répartition dans le temps de la manière dont cette énergie sera produite et consommée.

Nous pouvons utiliser les informations contenues dans le tableau *CostMin*, ainsi que dans le tableau *STATE^E*, en appliquant par exemple la formule suivante :

Soit $H = STATE_{0, E_0}^E$;

- $\text{Rough_Cost} = \text{CostMin}(0, H/3, 0) + \text{CostMin}(\lfloor N/3 \rfloor, H/3, 0) + \text{CostMin}(\lfloor 2N/3 \rfloor, H/3, 0);$
- $\beta = \text{Rough_Cost}/H.$

Cette formule est une sorte d'estimation statistique du coût énergétique par unité qu'il faut accepter pour que le véhicule puisse effectuer sa tournée.

6.7 Expérimentations numériques

6.7.1 Objectifs et contexte technique

Notre objectif est d'évaluer :

1. Le schéma **Pipe-line VD_PM** : nous nous concentrerons sur le compromis induit par ce schéma entre la précision (gap par rapport à l'optimalité) et le temps d'exécution (résumé par le nombre d'états générés tout au long du processus).
2. La puissance de la valeur du **Pipe-line VD_PM** comme borne supérieure pour l'algorithme général **DPS_SMEPC** décrits au chapitre 5. Nous nous concentrerons ici sur le nombre d'états générés au cours du processus en fonction des dispositifs de filtrage qui sont activés.

Les algorithmes ont été implémentés en C++ et l'IDE utilisé est Visual Studio 2017. Les expérimentations sont réalisées sur un ordinateur équipé d'un processeur AMD EPYC 7H12 64-Core, 512 Go de RAM et fonctionnent sous Gnu/linux Ubuntu 20.04.2. Le temps maximum du CPU est fixé à 1 heure.

6.7.2 Instances

Ici, nous utilisons un ensemble de 50 instances, avec les caractéristiques décrites par les tableaux (4.4) et (4.3).

Pour les instances A et B illustrées respectivement aux figures (5.7) et (5.10), on a illustré les solutions obtenues par l'algorithme **Pipe-line VD_PM**. Les solutions des instances A et B sont respectivement illustrées par les figures (6.2) et (6.3).

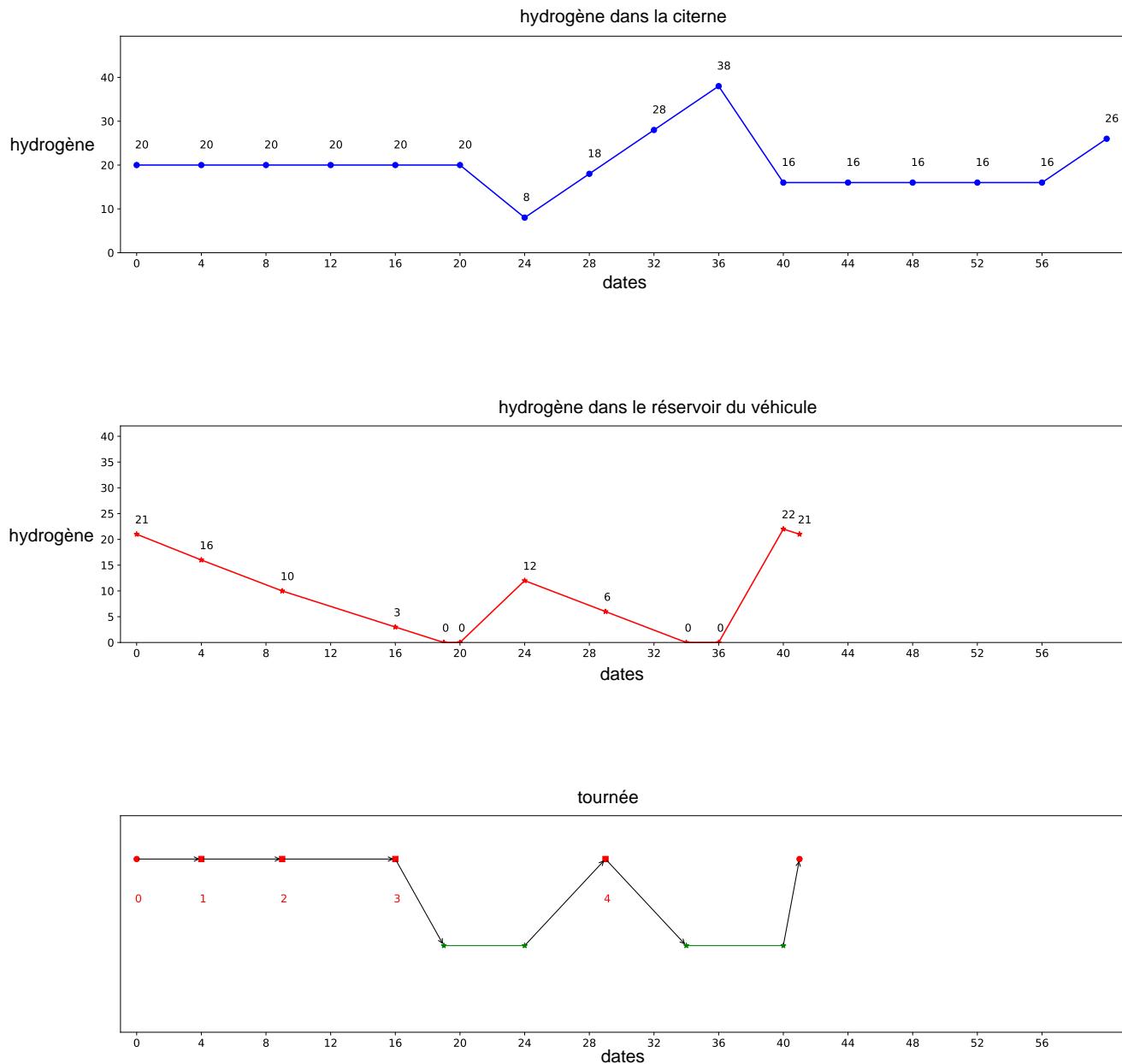


FIGURE 6.2 – Solution de l’instance A : on a deux recharges, l’une entre la station 3 et la station 4 et l’autre entre la station 4 et le dépôt final.

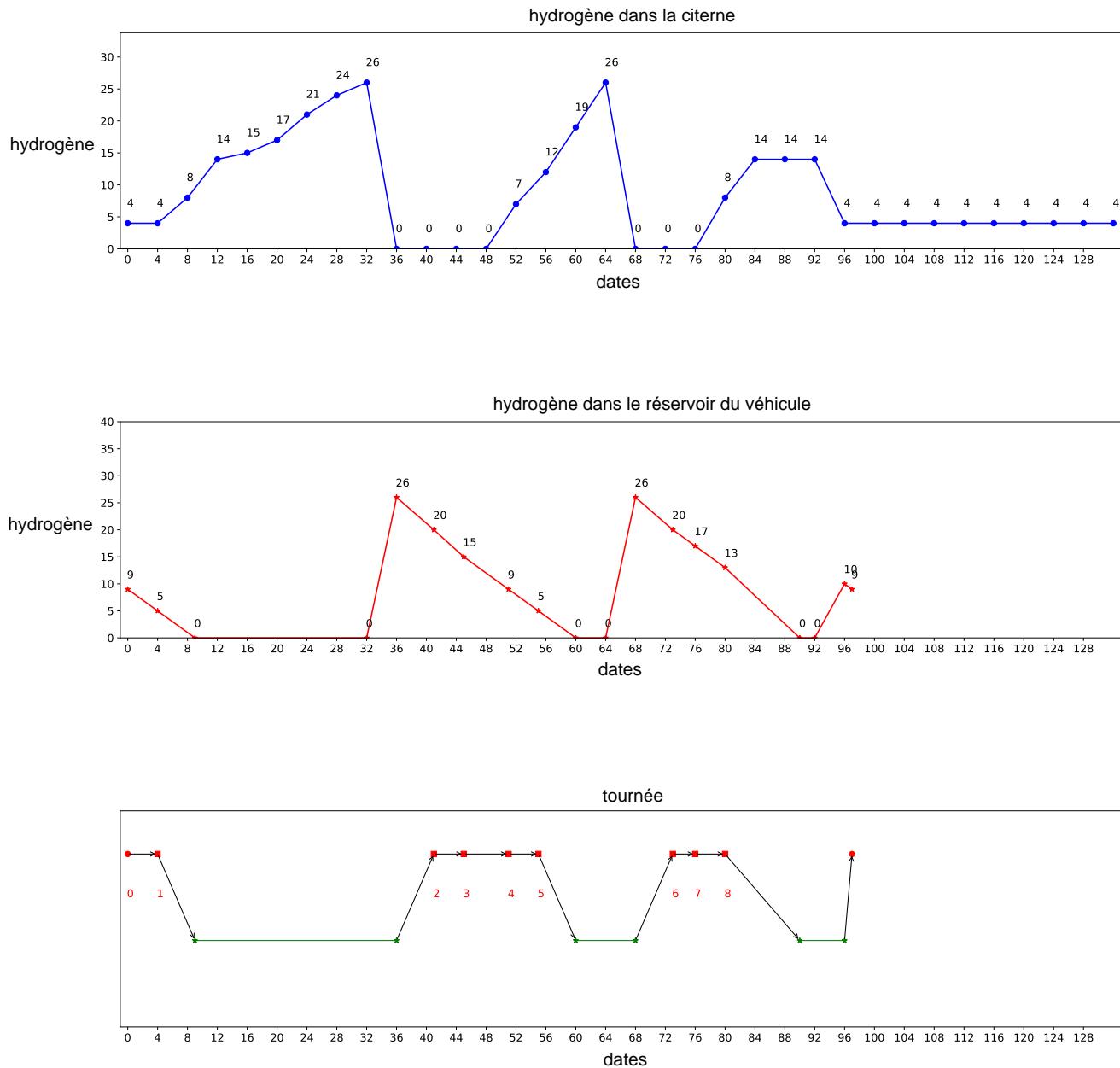


FIGURE 6.3 – Solution de l’instance B : on a trois recharges.

6.7.3 Comparaison de l’heuristique Pipe-line VD_PM et du DPS_SMEPC

Similairement à ce qui a été fait au chapitre précédent, on exécute l’algorithme Pipe-line VD_PM de quatre manières :

1. Sans mécanisme de filtrage, on nomme ces résultats $Pi(1)$;
2. Avec les mécanismes de filtrage logique, on nomme ces résultats $Pi(2)$;

3. Avec les mécanismes de filtrage logique et les mécanismes de filtrage par estimation optimiste, on nomme ces résultats $Pi(3)$;
4. Avec les mécanismes de filtrage logique, les mécanismes de filtrage par estimation optimiste et les mécanismes de filtrage heuristique, on nomme ces résultats $Pi(0)$. Concernant le filtrage heuristique, on a testé plusieurs valeurs de K et on a choisi la plus petite valeur qui permettait d'avoir une solution de chaque instance. Le paramètre obtenu pour ces instances est $K = 3$.

Les tableaux de résultats ont la signification suivante :

- Les tableaux (6.12) et (6.14) contiennent les gaps par rapport à l'optimalité et le temps CPU des algorithmes respectivement pour les paquets d'instances INST_CTE et INST_VAR. La signification des colonnes est :
 - **Obj** est la valeur obtenue ;
 - **Gap** est le gap par rapport à l'optimalité $Gap = 100 \times \frac{Obj - opt}{opt}$; où Obj est la valeur obtenue et opt est la valeur optimale obtenue à l'aide du modèle $MILP_{SMEPC}$. Si une instance n'a pas été résolu jusqu'à l'optimalité, on utilise la borne supérieure de l'exécution sur 8 threads avec un temps limite de 3 heures (voir tableau (4.7) et (4.8)).
 - **CPU** est le temps d'exécution en secondes.
- Les tableaux (6.13) et (6.15) présentent les résultats respectivement pour les paquets d'instances INST_CTE et INST_VAR. Les tableaux (6.13) et (6.15) contiennent le nombre maximal d'états Re , Pr et $\#Etats$, respectivement des algorithmes DPS_VD, DPS_PM et DPS_SMEPC.

On cherche à obtenir une évaluation de la qualité des solutions des algorithmes Pipe-line VD_PM et une évaluation de l'impact des mécanismes de filtrage. Pour cela, on exécute plusieurs fois l'algorithme **Pipe-line VD_PM** en activant à chaque fois les filtrages qui nous intéressent. La borne supérieure est calculée par l'algorithme 7 présenté à la section 5.6.3 du chapitre précédent.

Pour le paquet d'instances INST_CTE et le paquet d'instances INST_VAR, on fait la remarque suivante :似ilairement au chapitre précédent, lorsqu'on ajoute les mécanismes de filtrage l'algorithme est de plus en plus rapide car les mécanismes de filtrage diminuent le nombre d'états. On constate qu'il n'y a pas diminution des gaps lorsqu'on ajoute les mécanismes de filtrage logique. On fait les remarques suivantes :

- Comme pour le programme dynamique global, on a utilisé la même méthode de filtrage et les conclusions sont identiques. C'est-à-dire que l'ajout du filtrage heuristique est efficace. C'est majoritaire sur le paquet d'instances INST_VAR qu'on voit l'efficace du filtrage heuristique (on a une amélioration des gaps et une diminution du temps d'exécution des instances du paquet INST_VAR) ;
- Concernant les instances du paquet INST_CTE, l'algorithme Pipe-Line est plus rapide avec les filtrages heuristiques mais la qualité de la solution n'est pas impactée ;
- Le nombre d'états du paquet d'instances INST_CTE est inférieur au nombre d'états du paquet d'instances INST_VAR ;
- Le nombre d'états du Pipe-Line est inférieur à celui du programme dynamique ;
- Le gap et le temps CPU de l'heuristique Pipe-Line sont de moins bonne qualité que ceux de l'algorithme DPS_SMEPC avec tous les filtrages. Mais par contre le Pipe-Line réussit à résoudre certaines instances que le DPS_SMEPC avec les filtrages exactes ne parvient pas à résoudre.

Comparaison de l'heuristique Pipe-line VD_PM et du DPS_SMEPC : les instances du paquet INST_VAR

Les tableaux (6.12) et (6.13) présentent les résultats des instances du paquet INST_VAR. Lorsque la valeur vaut « - » cela veut dire que l'algorithme s'est arrêté au bout d'une heure sans trouver de solution.

Comparaison des gaps et du temps CPU des instances du paquet INST_VAR :

Le tableau (6.12) présente la comparaison des gaps et du temps de calcul (CPU) en secondes obtenus par l'exécution des algorithmes Pipe-line VD_PM sur les instances du paquet INST_VAR. En analysant les résultats des instances du paquet INST_VAR, on remarque que le filtrage heuristique est efficace car on obtient toujours une solution en moins d'une heure. Or, $Pi(1)$, $Pi(2)$ et $Pi(3)$ ne parvient pas à résoudre en moyenne 47,7% des instances en moins d'une heure. De plus, en ajoutant le filtrage heuristique la méthode devient très rapide et on obtient des gaps qui sont inférieurs à 7%, ce filtrage ne dégrade que légèrement la solution. De plus, avec le filtrage heuristique, on obtient 6 solutions optimales en moins de 0,01 seconde. Par exemple, pour l'instance 36, on passe de 28,8 minutes de temps d'exécution lorsqu'on exécute sans filtrage à 0,68 seconde lorsqu'on exécute avec tous les filtrages et on a un gap de 0,54%. Globalement le gap moyen des colonnes $Pi(0)$, $Pi(3)$, $Pi(2)$, $Pi(1)$ est respectivement 1,21 ; 1,21 ; 2,31 ; 2,31 et le temps d'exécution moyen en seconde est respectivement 210,78 ; 1665,32 ; 1887,92 ; 1924,61. On a calculé la moyenne des gaps seulement sur les 11 premiers instances.

	Pi(0)			Pi(3)			Pi(2)			Pi(1)		
	Tous les filtrages exacts et heuristiques			Tous les filtrages exacts			Filtrages logiques et optimistes			Sans filtrage		
num	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU
1	131	0,00	0,00	131	0,00	0,00	131	0,00	0,00	131	0,00	0,00
2	151	0,00	0,00	151	0,00	0,00	151	0,00	0,00	151	0,00	0,00
3	144	0,00	0,00	144	0,00	0,00	154	6,94	0,22	154	6,94	0,22
4	149	6,43	0,00	149	6,43	0,00	149	6,43	0,01	149	6,43	0,01
5	161	0,00	0,00	161	0,00	0,00	161	0,00	0,34	161	0,00	0,35
6	179	0,56	0,01	179	0,56	0,14	179	0,56	0,18	179	0,56	0,18
7	222	0,00	0,00	222	0,00	0,00	222	0,00	0,00	222	0,00	0,00
8	195	1,56	0,00	195	1,56	0,01	195	1,56	0,33	195	1,56	0,33
9	644	0,00	0,00	644	0,00	5,70	644	0,00	34,15	644	0,00	34,69
10	1142	0,26	0,00	1142	0,26	0,94	1142	0,26	1,91	1142	0,26	1,94
11	140	4,48	0,00	140	4,48	0,00	147	9,70	0,84	147	9,70	0,84
12	916	0,44	0,48	916	0,44	1670,67	-	-	3600,00	-	-	3600,00
13	960	0,42	0,06	960	0,42	36,50	960	0,42	193,63	960	0,42	193,78
14	1387	1,09	0,33	1387	1,09	3600,00	-	-	3600,00	-	-	3600,00
15	1348	0,15	0,34	1348	0,15	67,34	1348	0,15	77,18	1348	0,15	78,36
16	1298	0,54	0,68	1298	0,54	387,76	1298	0,54	1176,70	1298	0,54	1729,44
17	1318	2,57	5,35	-	-	3600,00	-	-	3600,00	-	-	3600,00
18	2390	0,72	0,84	2390	0,72	381,34	2390	0,72	1152,14	2390	0,72	1698,25
19	2306	4,20	67,40	-	-	3600,00	-	-	3600,00	-	-	3600,00
20	2679	0,75	0,34	2679	0,75	609,14	-	-	3600,00	-	-	3600,00
21	1496	6,10	63,50	-	-	3600,00	-	-	3600,00	-	-	3600,00
22	1365	1,71	63,40	-	-	3600,00	-	-	3600,00	-	-	3600,00
23	2743	5,91	281,64	-	-	3600,00	-	-	3600,00	-	-	3600,00
24	2943	4,92	88,85	-	-	3600,00	-	-	3600,00	-	-	3600,00
25	2124	3,46	4,11	-	-	3600,00	-	-	3600,00	-	-	3600,00
26	2376	0,85	4489,96	-	-	3600,00	-	-	3600,00	-	-	3600,00
27	2475	0,28	0,43	2475	0,28	3600,01	-	-	3600,00	-	-	3600,00
28	3138	2,75	415,68	-	-	3600,00	-	-	3600,00	-	-	3600,00
29	4506	1,56	780,54	-	-	3600,00	-	-	3600,00	-	-	3600,00
30	3618	2,73	59,33	-	-	3600,00	-	-	3600,00	-	-	3600,00

TABLE 6.12 – Les instances du paquet INST_VAR : Comparaison des gaps et du temps CPU du Pipe-Line VD_PM.

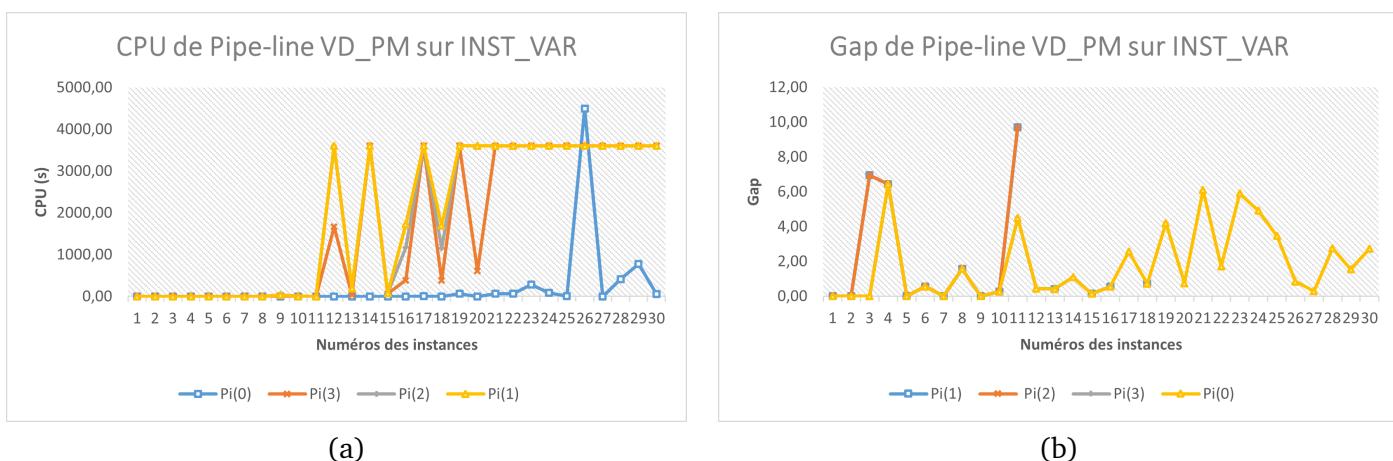


FIGURE 6.4 – Représentation graphique du tableau (6.12). (a) représente le temps CPU et (b) représente le gap de chaque instance de INST_VAR.

Comparaison du nombre d'états générés des instances du paquet INST_VAR :

Le tableau (6.13) présente la comparaison du nombre d'états générés par le Pipe-line VD_PM et le

DPS_SMEPC pour les instances du paquet INST_VAR. Le nombre maximal d'états est appelé Re , Pr et $\#Etats$, respectivement pour les algorithmes DPS_VD, DPS_PM et DPS_SMEPC. Le nombre d'états générés par l'algorithme DPS_VD est inférieur au nombre d'états générés par l'algorithme DPS_PM ce qui montre que le problème de production est le plus difficile des deux. Le nombre d'états générés par l'algorithme DPS_SMEPC est supérieur au nombre d'états générés par l'algorithme Pipe-Line VD_PM. Globalement le nombre d'états moyen de $Pi(0)$, $Pi(3)$, $Pi(2)$, $Pi(1)$ est respectivement 7703,667 ; 137301,267 ; 201852,033 ; 202426,6 et le nombre d'états moyen du DPS_SMEPC est respectivement 2449,133 ; 799762,733 ; 962697,333 ; 1033360,2.

num	Pi(0)			Pi(3)			Pi(2)			Pi(1)		
	Tous les filtrages exacts et heuristiques			Tous les filtrages exacts			Filtrages logiques et optimistes			Sans filtrage		
	Re	Pr	#Etats	Re	Pr	#Etats	Re	Pr	#Etats	Re	Pr	#Etats
1	3	42	61	3	78	554	3	669	4329	3	745	4604
2	3	84	161	3	381	1739	3	683	4241	3	683	4820
3	5	29	63	5	99	2213	5	5932	129720	5	5932	131744
4	3	142	321	3	748	2155	3	887	4748	3	887	4865
5	3	75	48	3	223	406	3	6880	82005	3	6880	83001
6	3	388	198	3	3142	9960	3	3275	13761	3	3275	19072
7	3	71	126	3	352	1732	3	531	7914	3	531	14375
8	3	160	87	3	1189	1607	3	6879	37060	3	6879	39376
9	4	200	286	4	15047	263867	4	33344	1410815	4	33344	1344462
10	3	128	148	3	8682	326094	3	10295	716815	3	10295	1024128
11	3	109	29	3	343	536	3	8681	42913	3	8681	43867
12	3	1343	314	3	167455	613249	3	292458	1020778	3	292458	1020778
13	4	575	59	4	34943	7115	4	75634	1610932	4	75634	1610932
14	3	990	642	3	191783	1550626	3	274485	1559487	3	314032	1559487
15	4	1485	106	4	46227	261078	4	48830	1067321	4	48830	1115578
16	5	1946	429	5	93550	909386	5	149187	1118045	5	149187	1118045
17	5	3264	410	5	223991	703595	5	279528	1019333	5	313384	1169832
18	5	1921	390	5	78975	872810	5	124484	764772	5	124484	1256433
19	4	10669	442	4	230526	2238483	4	349693	2010204	4	439033	2238483
20	6	1274	609	6	113612	1137757	6	379034	918477	6	379034	1227651
21	4	11650	2138	4	274431	1207299	4	352392	846485	4	352392	1219554
22	4	9253	5148	4	262745	1759894	4	347328	1687911	4	329060	1850181
23	4	21541	1284	4	321030	1320005	4	391639	1320005	4	374281	1320005
24	4	11074	1116	4	330588	1450961	4	386736	1450961	4	362386	1450961
25	8	1832	869	8	251611	1626287	8	417729	1484155	8	417729	1484155
26	6	94544	53759	6	197492	1550660	6	352015	1637832	6	352015	1637832
27	6	799	696	6	242065	1817157	6	433076	2510242	6	410623	2510242
28	4	18516	1258	4	309715	1468315	4	403883	1468854	4	388750	1468854
29	4	28811	1070	4	337806	1764386	4	428650	1764386	4	380630	1861070
30	8	8068	1207	8	380082	1122956	8	490597	1166419	8	490597	1166419

TABLE 6.13 – Les instances du paquet INST_VAR : impact des mécanismes de filtrage.

Comparaison de l'heuristique Pipe-line VD_PM et du DPS_SMEPC : les instances du paquet INST_CTE

Les tableaux (6.14) et (6.15) présentent les résultats des instances du paquet INST_CTE.

Comparaison des gaps et du temps CPU des instances du paquet INST_CTE :

Similairement à la section précédente, le tableau (6.14) présente la comparaison des gaps et la comparaison du temps CPU en secondes obtenus lors de l'exécution des algorithmes Pipe-line VD_PM sur les instances du paquet INST_CTE. En analysant les résultats des instances du paquet INST_CTE, on remarque qu'avec le filtrage heuristique on obtient 50% de solutions optimales très rapidement. On

remarque que le temps d'exécution des algorithmes Pipe-Line VD_PM est très rapide car il est toujours inférieur à 0,01 seconde. Concernant le gap, les filtrages sur les algorithmes Pipe-Line VD_PM sont moins efficaces que sur le schéma général DPS_SMEPC. Car on constate que pour 18 instances les gaps obtenus sans filtrage sont identiques à ceux avec filtrage. Par contre on remarque une amélioration du temps d'exécution. Globalement, le gap moyen de Pi(0), Pi(3), Pi(2), Pi(1) est respectivement 2,253 ; 2,253 ; 3,767 ; 3,767 et le temps d'exécution moyen en seconde est respectivement 0,00005 ; 0,00145 ; 0,04955 ; 0,0529.

	Pi(0)			Pi(3)			Pi(2)			Pi(1)		
	Tous les filtrages exacts et heuristiques			Tous les filtrages exacts			Filtrages logiques et optimistes			Sans filtrage		
num	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU	Obj	Gap	CPU
31	260	7,88	0,00	260	7,88	0,00	260	7,88	0,00	260	7,88	0,00
32	373	4,19	0,00	373	4,19	0,00	457	27,65	0,00	457	27,65	0,00
33	223	0,00	0,00	223	0,00	0,00	223	0,00	0,00	223	0,00	0,00
34	202	0,00	0,00	202	0,00	0,00	202	0,00	0,00	202	0,00	0,00
35	298	15,95	0,00	298	15,95	0,00	298	15,95	0,00	298	15,95	0,00
36	237	4,87	0,00	237	4,87	0,00	237	4,87	0,01	237	4,87	0,01
37	177	0,00	0,00	177	0,00	0,00	177	0,00	0,00	177	0,00	0,00
38	283	0,35	0,00	283	0,35	0,00	288	2,13	0,00	288	2,13	0,00
39	332	6,07	0,00	332	6,07	0,00	332	6,07	0,00	332	6,07	0,00
40	336	0,00	0,00	336	0,00	0,00	336	0,00	0,00	336	0,00	0,01
41	915	1,55	0,00	915	1,55	0,00	915	1,55	0,00	915	1,55	0,00
42	1440	0,00	0,00	1440	0,00	0,00	1440	0,00	0,00	1440	0,00	0,00
43	1568	2,08	0,00	1568	2,08	0,00	1568	2,08	0,00	1568	2,08	0,00
44	919	0,00	0,00	919	0,00	0,00	919	0,00	0,04	919	0,00	0,04
45	930	0,11	0,00	930	0,11	0,02	930	0,11	0,23	930	0,11	0,24
46	868	2,00	0,00	868	2,00	0,00	911	7,05	0,08	911	7,05	0,08
47	942	0,00	0,00	942	0,00	0,00	942	0,00	0,22	942	0,00	0,25
48	1250	0,00	0,00	1250	0,00	0,00	1250	0,00	0,07	1250	0,00	0,09
49	831	0,00	0,00	831	0,00	0,00	831	0,00	0,20	831	0,00	0,21
50	1080	0,00	0,00	1080	0,00	0,01	1080	0,00	0,12	1080	0,00	0,12

TABLE 6.14 – Les instances du paquet INST_CTE : comparaison des gaps et du temps CPU du Pipe-Line VD_PM.

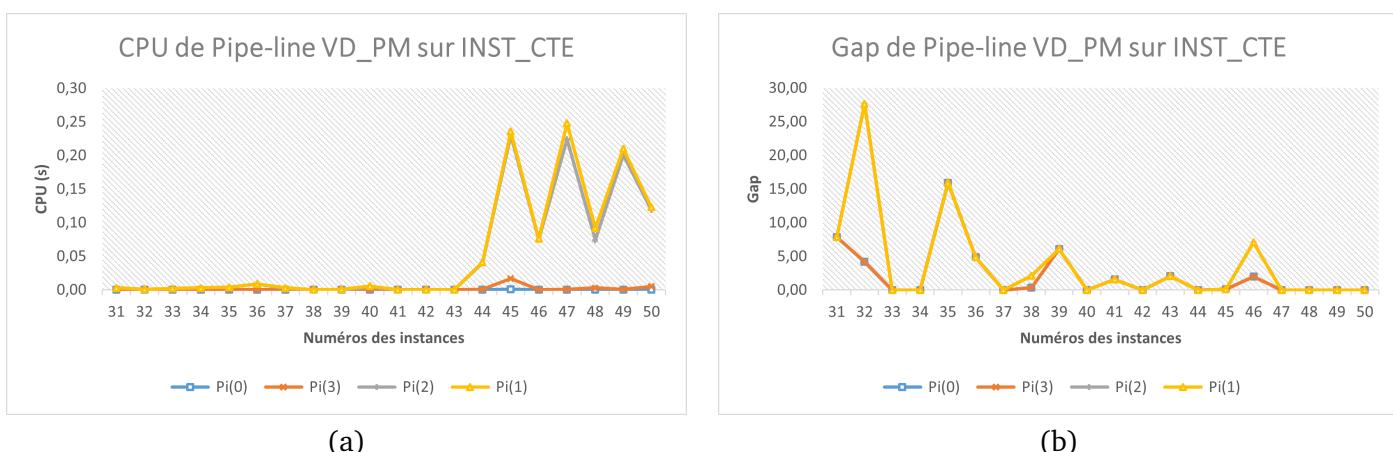


FIGURE 6.5 – Représentation graphique du tableau (6.14). (a) représente le temps CPU et (b) représente le gap de chaque instance de INST_CTE.

Comparaison du nombre d'états générés des instances du paquet INST_CTE :

Similairement à la section précédente, le tableau (6.15) présente la comparaison du nombre d'états générés par les algorithmes Pipe-line VD_PM et DPS_SMEPC pour les instances du paquet INST_CTE.

Le nombre maximal d'états est appelé Re , Pr et $\#Etats$, respectivement pour les algorithmes DPS_VD, DPS_PM et DPS_SMEPC. Le nombre d'états générés par l'algorithme DPS_VD est inférieur au nombre d'états générés par l'algorithme DPS_PM ce qui montre que le problème de production est le plus difficile. Le nombre d'états générés par l'algorithme DPS_SMEPC est supérieur au nombre d'états générés par l'algorithme Pipe-Line VD_PM. Globalement le nombre d'états moyen de $Pi(0)$, $Pi(3)$, $Pi(2)$, $Pi(1)$ est respectivement 65,85 ; 407,9 ; 2334 ; 2467,5 et le nombre d'états moyen de DPS_SMEPC est respectivement 118,2 ; 13746,85 ; 191125,9 ; 216886,55.

num	Pi(0)			Pi(3)			Pi(2)			Pi(1)		
	Tous les filtrages exacts et heuristiques			Tous les filtrages exacts			Filtrages logiques et optimistes			Sans filtrage		
	Re	Pr	#Etats	Re	Pr	#Etats	Re	Pr	#Etats	Re	Pr	#Etats
31	3	30	131	3	216	3077	3	689	10083	3	717	11422
32	5	9	80	5	9	1074	5	414	6042	5	489	6834
33	3	29	56	3	51	1670	3	578	11016	3	580	11504
34	4	36	120	4	255	5631	4	701	9757	4	701	11491
35	5	69	164	5	238	6728	5	908	20679	5	1030	23413
36	3	30	48	3	55	531	3	1429	67657	3	1429	68316
37	4	24	46	4	63	602	4	726	64720	4	890	66144
38	4	17	190	4	46	5513	4	187	11052	4	312	15478
39	6	15	110	6	54	8724	6	393	14777	6	520	18045
40	7	13	149	7	64	24125	7	993	314059	7	1393	327525
41	6	40	301	6	120	35121	6	198	62471	6	285	82386
42	7	44	227	7	155	16635	7	160	27568	7	239	40821
43	7	26	169	7	68	31508	7	68	31508	7	68	61652
44	5	117	75	5	650	4957	5	3296	117870	5	3296	131961
45	10	250	73	10	2329	8930	10	7338	221946	10	7338	240786
46	6	37	98	6	118	11841	6	4603	388485	6	4603	424646
47	10	93	72	10	578	14009	10	7204	841912	10	7717	890054
48	15	80	117	15	1066	64992	15	4045	662529	15	4941	893074
49	10	91	44	10	461	5497	10	7377	474977	10	7429	501320
50	10	137	94	10	1432	23772	10	5243	463410	10	5243	510859

TABLE 6.15 – Les instances du paquet INST_CTE : impact des mécanismes de filtrage.

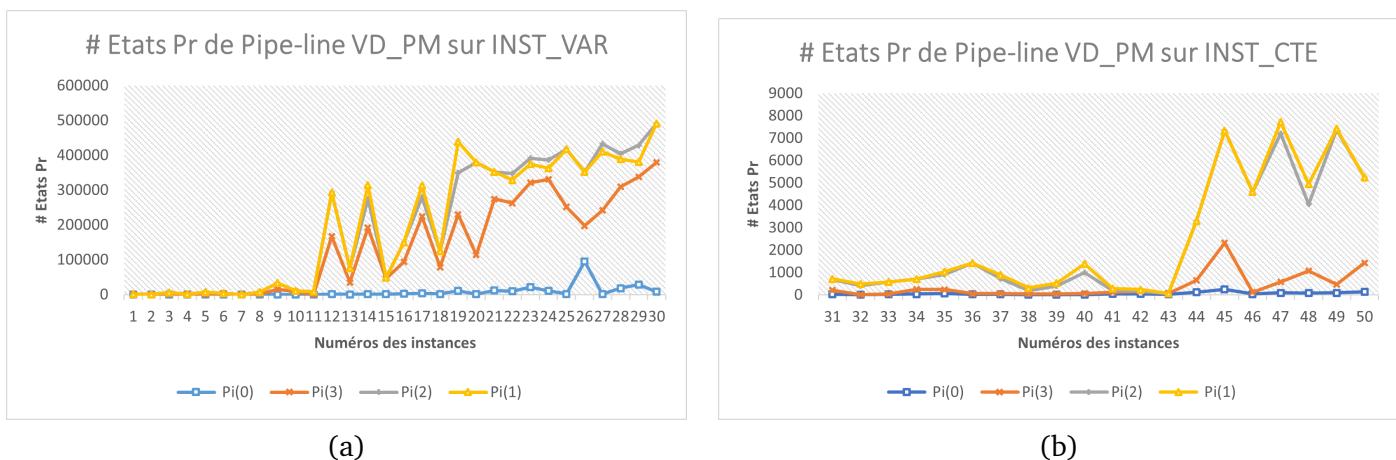


FIGURE 6.6 – Représentation graphique des tableaux (6.13) et (6.15). (a) représente le nombre d'états de Pipe-line VD_PM sur INST_VAR et (b) représente le nombre d'états de Pipe-line VD_PM sur INST_CTE.

6.7.4 Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC

Pour améliorer la procédure général **DPS_SMEPC**, on va utiliser la valeur de l'algorithme **Pipe-line VD_PM** comme borne supérieure. Dans cette section, on présente les nouveaux résultats du **DPS_SMEPC** qu'on a obtenu avec cette nouvelle borne supérieure. Notre objectif ici est de savoir si l'utilisation de la valeur de l'algorithme **Pipe-line VD_PM** comme borne supérieure améliore l'algorithme **DPS_SMEPC**. On cherche à savoir s'il y amélioration du gap par rapport à l'optimalité et s'il y a diminution du nombre d'états générés. Pour cela, on exécute plusieurs fois l'algorithme **DPS_SMEPC** en activant à chaque fois les filtrages qui nous intéressent. La modification de la borne supérieure a un impact uniquement sur $St(0)$ et $St(3)$. On ne calcule pas le gap de $St(3)$ car on obtient la valeur optimale donc le gap vaut zéro. Les tableaux (6.18) et (6.19) présentent les résultats de l'utilisation de l'algorithme **Pipe-line VD_PM** comme borne supérieure du **DPS_SMEPC**. En analysant les résultats des instances, on remarque que le filtrage heuristique est efficace car en ajoutant ce filtrage la méthode devient très rapide et de plus les gaps sont inférieurs à 5%, ces filtrages ne dégradent que légèrement la solution.

Les tableaux (6.16) et (6.17) comparent les deux bornes supérieures utilisées pour le filtrage respectivement pour le paquet d'instances **INST_CTE** et le paquet d'instances **INST_VAR**. « Search BSUP » est la borne supérieure calculée par l'algorithme (7) présenté au chapitre précédent à la section 5.6.3. Et « $Pi(3)$ » est la nouvelle borne supérieure obtenue à l'aide de l'algorithme **Pipe-line VD_PM**. Globalement, pour les instances pour lesquelles les deux algorithmes donnent des solutions, on remarque que la colonne « $Pi(3)$ » donne une meilleure borne supérieure que la colonne « Search BSUP ». Particulièrement, pour onze instances du paquet d'instances **INST_VAR**, « $Pi(3)$ » ne trouve pas de solutions en moins d'une heure.

num	Search BSUP		Pi(3)	
	Obj	Gap	Obj	Gap
1	131	0,00	131	0,00
2	155	2,65	151	0,00
3	144	0,00	144	0,00
4	155	10,71	149	6,43
5	161	0,00	161	0,00
6	199	11,80	179	0,56
7	222	0,00	222	0,00
8	200	4,17	195	1,56
9	680	5,59	644	0,00
10	1178	3,42	1142	0,26
11	140	4,48	140	4,48
12	970	6,36	916	0,44
13	970	1,46	960	0,42
14	1513	10,28	1387	1,09
15	1371	1,86	1348	0,15
16	1361	5,42	1298	0,54
17	1318	2,57	-	-
18	2490	4,93	2390	0,72
19	2449	10,66	-	-
20	2733	2,78	2679	0,75
21	1529	8,44	-	-
22	1465	9,17	-	-
23	2837	9,54	-	-
24	3058	9,02	-	-
25	2147	4,58	-	-
26	2459	4,37	-	-
27	2543	3,04	2475	0,28
28	3213	5,21	-	-
29	4740	6,83	-	-
30	3621	2,81	-	-

TABLE 6.16 – Les instances du paquet INST_VAR : Comparaison des deux bornes supérieures utilisées pour le filtrage du DPS_SMEPC.

num	Search BSUP		Pi(3)	
	Obj	Gap	Obj	Gap
31	325	34,85	260	7,88
32	373	4,19	373	4,19
33	223	0,00	223	0,00
34	235	16,34	202	0,00
35	298	15,95	298	15,95
36	237	4,87	237	4,87
37	177	0,00	177	0,00
38	283	0,35	283	0,35
39	339	8,31	332	6,07
40	336	0,00	336	0,00
41	924	2,55	915	1,55
42	1440	0,00	1440	0,00
43	1623	5,66	1568	2,08
44	919	0,00	919	0,00
45	947	1,94	930	0,11
46	868	2,00	868	2,00
47	942	0,00	942	0,00
48	1325	6,00	1250	0,00
49	831	0,00	831	0,00
50	1084	0,37	1080	0,00

TABLE 6.17 – Les instances du paquet INST_CTE : Comparaison des deux bornes supérieures utilisées pour le filtrage du DPS_SMEPC.

Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC : les instances du paquet INST_VAR

Le tableau (6.18) présente les résultats des instances du paquet INST_VAR. Lorsque la valeur vaut « - » cela veut dire que l'algorithme s'est arrêté au bout d'une heure sans trouver de solution. La colonne $St(3)$ montre que l'algorithme DPS_SMEPC ne parvient pas à résoudre 50% des instances en moins d'une heure. De plus avec le filtrage heuristique on obtient 9 solutions optimales en moins de 0,4 seconde. Par exemple, pour l'instance 16, on passe de 7,55 minutes de temps d'exécution lorsqu'on exécute sans filtrage heuristique à 0,08 seconde lorsqu'on exécute avec tous les filtrages et on a un gap de 0,08%. La procédure DPS_SMEPC avec tous les filtrages exactes ($St(3)$) ne parvenait pas à résoudre l'instance 12 en moins d'une heure, or en utilisant la borne supérieure de Pipe-line VD_PM elle y parvient maintenant en 0,03 seconde. En moyenne sans filtrage heuristique on a généré 629592,9 états et en ajoutant le filtrage heuristique on passe à 1110,767 états en moyenne. Ce qui nous permet d'affirmer que ce filtrage est efficace. Le gap moyen de $St(0)$ est 0,23%. Le nombre d'états moyen de $St(0)$ et $St(3)$ est respectivement 1110,767 et 629592,9. Le temps d'exécution moyen en minutes de $St(0)$ et $St(3)$ est respectivement 0,16 et 31,25. Dans la section 5.6.4 du chapitre précédent, le nombre d'états moyen de $St(0)$ et $St(3)$ était respectivement 2449,133 ; 799762,7333 et, le temps d'exécution moyen en minutes était respectivement 0,96 ; 36,1048. De plus, le gap moyen de $St(0)$ était 0,596%. En comparant le tableau (6.18) à celui de la section 5.6.4, on constate qu'il y a une diminution du gap, du temps d'exécution et du nombre d'états générés par l'algorithme DPS_SMEPC. Ce qui prouve que l'utilisation de la valeur du Pipe-Line VD_PM comme borne supérieure a amélioré la procédure DPC_SMEPC.

num	St(0)				St(3)		
	Tous les filtrages exacts et heuristiques				Tous les filtrages exacts		
	Obj	Gap	#Etats	CPU	Obj	#Etats	CPU
1	131	0,00	61	0,00	131	554	0,00
2	151	0,00	159	0,00	151	1336	0,01
3	144	0,00	63	0,00	144	2213	0,03
4	140	0,00	281	0,01	140	1844	0,02
5	161	0,00	48	0,00	161	406	0,01
6	179	0,56	78	0,00	178	2472	0,04
7	222	0,00	126	0,00	222	1732	0,01
8	192	0,00	72	0,00	192	965	0,01
9	644	0,00	161	0,01	644	6962	0,18
10	1142	0,26	102	0,01	1139	49609	2,37
11	140	4,48	29	0,00	134	536	0,01
12	914	0,22	176	0,03	912	12756	1,67
13	960	0,42	58	0,00	956	3391	0,15
14	1372	0,00	460	0,31	-	754539	3600,43
15	1348	0,15	88	0,02	1346	88677	22,87
16	1292	0,08	230	0,08	1291	327838	453,14
17	1288	0,23	410	0,73	-	900272	3615,24
18	2383	0,42	257	0,20	-	939836	3740,62
19	2218	0,23	724	2,13	-	1908524	3669,73
20	2655	-0,15	498	1,00	-	1180949	3631,16
21	1412	0,14	2037	17,51	-	1216931	3626,03
22	1350	0,60	1086	3,33	-	773791	3600,59
23	2596	0,23	745	2,53	-	1076713	3805,69
24	2807	0,07	629	2,25	-	1443810	3949,79
25	2060	0,34	1052	8,05	-	1683461	3626,42
26	2353	-0,13	18731	205,21	-	817959	3655,21
27	2475	0,28	488	1,39	-	704556	3606,82
28	3066	0,39	1997	13,72	-	1357220	4288,92
29	4377	-1,35	1122	4,11	-	1861070	3707,21
30	3500	-0,62	1355	20,32	-	1766865	3646,29

TABLE 6.18 – Les instances du paquet INST_VAR : impact des mécanismes de filtrage du DPS_SMEPC.

Utilisation de l'algorithme Pipe-line VD_PM comme borne supérieure du DPS_SMEPC : les instances du paquet INST_CTE

Le tableau (6.19) présente les résultats des instances du paquet INST_CTE. En ajoutant le filtrage heuristique, on obtient 70% de solutions optimales. Par exemple, pour l'instance 48, on passe de 15,54 secondes de temps d'exécution lorsqu'on exécute sans filtrage heuristique à 0,01 seconde lorsqu'on exécute avec le filtrage heuristique et on obtient la solution optimale. En moyenne sans filtrage heuristique, on a généré 12489,1 états et en ajoutant le filtrage heuristique on passe à 112,75 en moyenne d'états générés. Ce qui nous permet d'affirmer une fois de plus que ce filtrage est efficace. Le gap moyen de St(0) est 0,562%. Le nombre d'états moyen de St(0) et St(3) est respectivement 112,75 et 12489,1. Le temps d'exécution moyen en secondes de St(0) et St(3) est respectivement 0,0068 et 1,54. Dans la section 5.6.4 du chapitre précédent, le nombre d'états moyen de St(0), St(3) était respectivement 118,2 ; 13746,85 et le temps d'exécution moyen en secondes était respectivement 0,007 ; 2,448. De plus le gap moyen de St(0) était 1,108%. En comparant le tableau (6.19) à celui de la section 5.6.4, on constate qu'il y a une diminution du gap, du temps d'exécution et du nombre d'états générés par l'algorithme DPS_SMEPC. Comme à la section précédente, ceci prouve que l'utilisation de la valeur du Pipe-Line VD_PM comme borne supérieure a amélioré la procédure DPS_SMEPC.

num	St(0)				St(3)		
	Tous les filtrages exacts et heuristiques				Tous les filtrages exacts		
	Obj	Gap	#Etats	CPU	Obj	#Etats	CPU
31	241	0,00	115	0,00	241	2341	0,02
32	373	4,19	80	0,00	358	1074	0,01
33	223	0,00	56	0,00	223	1670	0,01
34	202	0,00	96	0,00	202	3058	0,04
35	257	0,00	164	0,01	257	6728	0,11
36	226	0,00	48	0,00	226	531	0,01
37	177	0,00	46	0,00	177	602	0,01
38	282	0,00	190	0,01	282	5513	0,06
39	323	3,19	100	0,00	313	7397	0,06
40	336	0,00	149	0,01	336	24125	0,68
41	915	1,55	276	0,02	901	34026	3,56
42	1440	0,00	227	0,02	1440	16635	1,28
43	1568	2,08	169	0,01	1536	31508	2,75
44	919	0,00	75	0,00	919	4957	0,19
45	930	0,11	63	0,00	929	7534	0,49
46	852	0,12	98	0,02	851	11841	1,11
47	942	0,00	72	0,01	942	14009	1,48
48	1250	0,00	93	0,01	1250	48374	15,54
49	831	0,00	44	0,01	831	5497	0,51
50	1080	0,00	94	0,01	1080	22362	2,89

TABLE 6.19 – Les instances du paquet INST_CTE : impact des mécanismes de filtrage du DPS_SMEPC.

6.8 Conclusion

Nous avons présenté un schéma de programmation dynamique collaboratif afin de résoudre le problème SMEPC qui nécessite des mécanismes de synchronisation. Comme nous l'avons dit au début de ce chapitre, notre principal objectif ici est de mettre en œuvre la synchronisation tout en émulant les interactions qui sont susceptibles d'avoir lieu entre les acteurs décentralisés. Le mécanisme de pipeline à sens unique qu'on a implémenté est clairement l'un des plus simples. Mais on pourrait penser à concevoir des schémas d'interaction plus sophistiqués, et même si nous nous limitons à ce schéma spécifique, certains points peuvent être discutés : sur la simple structure de l'algorithme **Pipeline VD_PM**, le lien entre **VD** et **PM** défini par β se comporte ici comme un lien monodirectionnel : On attribue une valeur à β , et on lance successivement **DPS_VD** et **DPS_PM**. Mais on peut se demander comment récupérer des informations de cette séquence **DPS_VD** → **DPS_PM** et s'adapter en conséquence. Par exemple, on peut essayer ceci : après chaque exécution de la séquence **DPS_VD** → **DPS_PM**, attribuer à β une nouvelle valeur β^* telle que la quantité $\beta^* \times (\sum_j L_j \times x_j)$ devienne égale au coût énergétique $\sum_{i=0, \dots, N-1} (Cost^F \times y_i + Cost^V_i \times z_i)$ et recommencer le processus. Nous n'avons pas essayé cette option, principalement parce qu'elle prend beaucoup de temps. De la même manière, on pourrait faire en sorte que le coefficient α qui pondère les temps d'attente que le conducteur du véhicule doit accepter fasse partie du processus de négociation.

6.9 Annexes

6.9.1 Démonstration du théorème 5

Cette section présente la démonstration du théorème 5.

Dans cette partie nous allons démontrer que le problème VD avec L_j fixe est un problème NP-Hard. Nous allons montrer que le problème VD avec L_j fixe se réduit en temps polynomial au problème du sac à dos inversé. La formulation du problème du sac à dos (K) est la suivante : « Étant donnés n objets possédant chacun un poids p_i et une valeur v_i , $i = \{1, \dots, n\}$. Étant donné un sac de poids maximum b , quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal du sac ? » La modélisation de (K) sous forme de programme linéaire en nombres entiers est la suivante :

Soit un sac à dos de poids maximal b et n objets. Pour chaque objet i , nous avons un poids $p_i \geq 0$ et une valeur $v_i \geq 0$,

- La variable de décisions est y_i avec $i \in \{1, 2, \dots, n\}$

$$y_i = \begin{cases} 1 & \text{si l'objet } i \text{ est mis dans le sac} \\ 0 & \text{sinon.} \end{cases}$$

- La contrainte est $\sum_{i=1}^n p_i \times y_i \leq b$ car le sac à dos à un poids maximum b qu'il ne faut pas dépasser ;
- La fonction objectif à maximiser est $\sum_{i=1}^n v_i \times y_i$ car l'objectif est de maximiser la valeur totale des objets introduit dans le sac à dos.

Nous devons faire apparaître le problème du sac à dos inversé comme un cas particulier du problème VD avec L_j fixe. Pour cela, à une instance (n, P_i, v_i, b) du problème du sac à dos inversé, on doit associer à l'aide de formules une instance $(M, t_j, d_j, d_j^*, e_j, \varepsilon_j, \varepsilon_j^*, R, E_0, C^{Veh})$ du problème VD avec L_j fixe dont la signification est : M est le nombre de stations, les distances entre les stations sont (t_j, d_j, d_j^*) , les quantités d'énergies dépensées lors des trajets sont $(e_j, \varepsilon_j, \varepsilon_j^*)$, la quantité d'hydrogène chargée est (R) , l'énergie initiale du véhicule est (E_0) et le capacité maximal du réservoir d'hydrogène est (C^{Veh}) .

Une fois ces formules obtenues :

1. on veut montrer que $(M, t_j, d_j, d_j^*, e_j, \varepsilon_j, \varepsilon_j^*, R, E_0, C^{Veh})$ nous donne bien une instance du problème VD avec L_j fixe, et donc qu'on a t_j, d_j, d_j^* et $e_j, \varepsilon_j, \varepsilon_j^*$ qui sont des distances qui respectent l'inégalité du triangle. De plus, il faut que $t_j \geq 0, d_j \geq 0, d_j^* \geq 0$ et $e_j \geq 0, \varepsilon_j \geq 0, \varepsilon_j^* \geq 0$.
2. on veut montrer qu'une solution optimale du problème du sac inversé à dos s'interprète comme une solution optimale du problème VD avec L_j fixe.

Pour faire apparaître les stations, on pose $n = M$, $y_i = 1 - x_i$, $i = 1, \dots, n$ avec $x_i \in \{0, 1\}$, (K) devient :

$$\begin{cases} \text{Maximiser : } \sum_{i=1}^n v_i \times (1 - x_i) = \sum_{i=1}^n v_i - \sum_{i=1}^n v_i \times x_i \\ \text{st : } -\sum_{i=1}^n p_i \times x_i \leq -\sum_{i=1}^n p_i + b \end{cases}$$

Car $(1 - x_i) \in \{0, 1\} \Rightarrow x_i \in \{0, 1\}$

c'est-à-dire,

$$\begin{cases} \text{Minimiser : } -\sum_{i=1}^n v_i + \sum_{i=1}^n v_i \times x_i \\ \text{st : } \sum_{i=1}^n p_i \times x_i \geq (\sum_{i=1}^n p_i) - b \end{cases}$$

Car $Z_{\max}(X) = -Z_{\min}(-X)$

En ce qui concerne la capacité du réservoir d'hydrogène nous allons considérer qu'on a un véhicule dont la capacité du réservoir est très grande. Dans ce cadre là, l'objectif est de minimiser $\sum_{i=1}^n (d_i + d_{i+1}^* - t_i) \times x_i$ sous les contraintes (6.1), (6.2) et (6.3).

$$E_0 - \varepsilon_1^* - (\varepsilon_1 \times x_1) \geq 0 \quad (6.1)$$

$$\sum_{j=1}^i [(R + e_j - (\varepsilon_j + \varepsilon_{j+1}^*)) \times x_j] - (\varepsilon_{i+1} \times x_{i+1}) \geq \varepsilon_1^* + \sum_{j=1}^i e_j - E_0 \quad (6.2)$$

avec $i \in 1 \dots n - 1$

$$\sum_{i=1}^n [(R + e_i - (\varepsilon_i + \varepsilon_{i+1}^*)) \times x_i] \geq \varepsilon_1^* + \sum_{i=1}^n e_i + \varepsilon_n - E_0 \quad (6.3)$$

On pose $v_0 = p_0 = v_{n+1} = p_{n+1} = 0$. Pour obtenir les distances entre les stations et les quantités d'énergies dépensées lors des trajets, posons :

- $t_i = \frac{v_{i+1} + v_{i-1}}{2}$, $d_i^* = d_i = \frac{v_{i+1} + v_{i-1}}{2}$, pour $i = 1, \dots, n$
- $e_i = R - \frac{p_{i+1} + p_{i-1}}{2}$, $\varepsilon_i^* = \varepsilon_i = R - \frac{p_i + p_{i-1}}{2}$, pour $i = 1, \dots, n$,

avec la quantité d'hydrogène chargée R qui vaut $R = \max(p_i)$.

- La quantité d'hydrogène initialement dans le véhicule est :

$$E_0 = \varepsilon_1^* + \sum_{j=1}^n e_j + \max_{1 \leq i \leq n}(\varepsilon_i)$$

$$\varepsilon_n = (\sum_{i=1}^n p_i) - b + \max_{1 \leq i \leq n}(\varepsilon_i)$$

Les p_i et v_i peuvent être quelconques mais les t_j, d_j, d_j^* et $e_j, \varepsilon_j, \varepsilon_j^*$ doivent respecter l'inégalité triangulaire parce qu'elles doivent correspondre à des distances, nous vérifions alors que :

1. $d_i + d_{i+1}^* - t_i = \frac{v_{i+1} + v_{i-1}}{2} + \frac{v_{i+1} + v_i}{2} - \frac{v_{i+1} + v_{i-1}}{2} = v_i$, pour $i = 1, \dots, n$
donc $t_i \leq d_i + d_{i+1}^*$
2. $d_{i+1}^* + t_i - d_i = \frac{v_{i+1} + v_i}{2} + \frac{v_{i+1} + v_{i-1}}{2} - \frac{v_{i+1} + v_{i-1}}{2} = v_{i+1}$, pour $i = 1, \dots, n$
donc $d_i \leq t_i + d_{i+1}^*$
3. $t_i + d_i - d_{i+1}^* = \frac{v_{i+1} + v_{i-1}}{2} + \frac{v_{i+1} + v_{i-1}}{2} - \frac{v_{i+1} + v_i}{2} = v_{i-1}$, pour $i = 1, \dots, n$
donc $d_{i+1} \leq t_i + d_i^*$
4. $\varepsilon_i + \varepsilon_{i+1}^* - e_i = R - \frac{p_i + p_{i-1}}{2} + R - \frac{p_{i+1} + p_i}{2} - R + \frac{p_{i+1} + p_{i-1}}{2} = R - p_i$, pour $i = 1, \dots, n$
donc $e_i \leq \varepsilon_i + \varepsilon_{i+1}^*$ car $R = \max(p_i)$
5. $\varepsilon_i + e_i - \varepsilon_{i+1}^* = R - \frac{p_i + p_{i-1}}{2} + R - \frac{p_{i+1} + p_{i-1}}{2} - R + \frac{p_{i+1} + p_i}{2} = R - p_{i-1}$, pour $i = 1, \dots, n$
donc $\varepsilon_{i+1}^* \leq \varepsilon_i + e_i$ car $R = \max(p_i)$
6. $\varepsilon_{i+1}^* + e_i - \varepsilon_i = R - \frac{p_{i+1} + p_i}{2} + R - \frac{p_{i+1} + p_{i-1}}{2} - R + \frac{p_i + p_{i-1}}{2} = R - p_{i+1}$, pour $i = 1, \dots, n$
donc $\varepsilon_i \leq \varepsilon_{i+1}^* + e_i$ car $R = \max(p_i)$

$$R + e_i - (\varepsilon_i + \varepsilon_{i+1}^*) = 2R - \frac{p_{i+1} + p_{i-1}}{2} - 2R + \frac{p_i + p_{i-1}}{2} + \frac{p_{i+1} + p_i}{2} = p_i, \text{ pour } i = 1, \dots, n$$

et le second membre de (6.3) devient,

$$\varepsilon_1^* + \sum_{j=1}^n e_j - E_0 + \varepsilon_n = (\sum_{i=1}^n p_i) - b$$

et les seconds membres de (6.2) sont tous négatifs ou nuls car

$$E_0 \geq \varepsilon_1^* + \sum_{j=1}^i e_j + \varepsilon_{i+1}.$$

Il reste à vérifier l'inéquation (6.3). $\sum_{j=1}^i [(R + e_j - (\varepsilon_j + \varepsilon_{j+1}^*)) \times x_j] \geq \varepsilon_1^* + \sum_{j=1}^i e_j - E_0$ est aussi vrai avec $i = \{2, \dots, n\}$ On a donc démontré que résoudre le problème VD avec L_j fixe revient donc à résoudre le problème (K).

6.9.2 Démonstration du théorème 6

Cette section présente la démonstration du théorème 6. Dans cette section, nous allons prouver que **VD** est polynomiale. Cependant, nous sommes toujours à la frontière entre P et NP.

Le graphe **VD_Graph** : Afin de traiter le problème **VD**, nous allons faire apparaître qu'il peut être traitée comme un simple problème de plus court chemin dans le graphe orienté nommé **VD_Graph** suivant :

1. Les nœuds de **VD_Graph** sont :

- Une nœud source s et un nœud destination $p = ((M + 1), E_0)$;
- Les paires $(0, V)$, $0 \leq V \leq E_0$;
- Les paires (j, V) , $j = 1, \dots, M$, où V est un nombre non négatif, pas plus grand que C^{Veh} ;

2. Les arcs de **VD_Graph** sont :

- Tout arc $u = (s, (0, V))$, avec un coût négatif $D_u = (V - E_0)$; a
- Tout arc $u = ((j, V), (j + 1, V - e_j))$, avec un coût nul D_u ;
- Tout arc $u = ((j, \varepsilon_j), (j + 1, V))$, $V \geq \varepsilon_{j+1}$ avec un coût $D_u = \alpha \times (d_j + d_{j+1}^* - t_j + p) + \beta \times (\varepsilon_j + \varepsilon_{j+1}^* - e_j)$.

Le sens de cette construction se dégage à la suite du Lemme 3.

Lemme 3 *Pour résoudre **VD**, il faut chercher le plus court chemin de s à p dans **VD_Graph**.*

Démonstration : Nous disons que la **stratégie de recharge optimale** (x, L) satisfait l'hypothèse du réservoir vide si :

- Chaque fois que le véhicule fait le plein, mais pas la première fois, il arrive à la micro-usine avec un réservoir vide ;
- Le véhicule après sa tournée revient à la micro-usine avec une charge d'hydrogène exactement égale à E_0 .

Nous remarquons alors qu'une *stratégie de recharge optimale* (x, L) peut être choisie de manière à satisfaire à cette hypothèse de réservoir vide. Si (x, L) ne correspond pas à l'hypothèse du réservoir vide, et s'il arrive à la micro-usine avec une charge non nulle $\delta > 0$, alors il est possible de diminuer l'ancienne opération de recharge en carburant de δ' , et d'augmenter l'opération de recharge actuelle de $\delta' \leq \delta$, jusqu'à l'annulation de l'opération de recharge correspondante pour en faire un réservoir vide. C'est comme ça que :

- Les nœuds du **VD_Graph** correspondent aux états possibles du véhicule lorsqu'il effectue son trajet du dépôt initial $Depot = 0$ au dépôt final $Depot = M + 1$, tout en visitant les stations $j = 1, \dots, M$ et en faisant le plein périodiquement. Si le véhicule n'a jamais été rechargé en carburant avant j , $V \leq E_0$ signifie la quantité d'hydrogène qu'il va consommer avant la recharge ; sinon, il signifie la charge d'hydrogène actuelle du véhicule à j .
- C'est en fonction de cela que nous comprenons la signification des arcs :
 - L'arc $u = ((j, V), (j + 1, V - e_j))$, avec un coût nul D_u , signifie que le véhicule se déplace directement de j à $j + 1$;

- L'arc $u = ((j, \varepsilon_j), (j + 1, V))$, $V \geq \varepsilon_{j+1}$ avec un coût $D_u = \alpha \times (d_j + d_{j+1}^* - t_j + p) + \beta \times (\varepsilon_j + \varepsilon_{j+1}^* - e_j)$, signifie que le véhicule se déplace de j à $j + 1$ tout en faisant le plein de carburant à la micro-usine : le coût d'un tel déplacement est le coût supplémentaire (temps + énergie) induit par ce détour ;
- L'arc $u = (s, (0, V))$, avec $V \leq E_0$, avec un coût négatif $D_u = (V - E_0)$, signifie la décision que le véhicule va prendre au début du processus, lorsqu'il va décider de sa première opération de recharge en carburant : le trajet correspondant de 0 (le dépôt initial) jusqu'à $5 = 0$ (le dépôt final) va alors nécessiter V unités d'hydrogène, avec un coût négatif $D_u = (V - E_0)$ qui correspond au fait que le véhicule arrive à la micro-usine avec une surcharge $(E_0 - V)$ qui sera utilisée ultérieurement.

Nous en déduisons que toute **stratégie de recharge optimale** (x, L) qui satisfait l'hypothèse du réservoir vide correspond à un cheminement dans **VD_Graph**, dont le coût est exactement le coût de la **stratégie de recharge optimale** (x, L) , et inversement.

En fait, nous n'avons pas besoin de tout le graphe **VD_Graph** pour traiter **VD**. Si nous appliquons un algorithme de Bellman en *backward*, nous voyons que nous ne traitons que la collection de noeuds suivante $X(j), j = 0, \dots, M + 1$, dont la définition récursive est la suivante :

- $\square X(M + 1)$ est réduit à $p = ((M + 1), E_0)$ car le véhicule finit sa tournée au dépôt final $M + 1$ avec une quantité d'hydrogène dans son réservoir égale à E_0 ;
- $\square X(M) = \{(M, \varepsilon_M), (M, E_0 + e_M)\}$ car lorsque le véhicule quitte la station M il a deux choix :
 - soit le véhicule va se recharger auquel cas il a besoin d'une quantité d'hydrogène ε_M pour arriver à la micro-usine ;
 - soit le véhicule se rend directement au dépôt final auquel cas il a besoin d'une quantité d'hydrogène e_M pour y arriver et E_0 pour respecter les contraintes du problème **VD**.
- \square Pour $j = 0, \dots, M - 1$, $X(j) = \{(j, \varepsilon_j)\} \cup \{(j, V + e_j), \forall (j+1, V) \in X(j+1)\}$ tel que $V + e_j \leq C^{Veh}$. Car lorsque le véhicule quitte la station j il a deux choix :
 - soit le véhicule va se recharger auquel cas il a besoin d'une quantité d'hydrogène ε_j pour arriver à la micro-usine ;
 - soit le véhicule se rend directement à la station suivante $j + 1$ auquel cas il a besoin d'une quantité d'hydrogène e_j pour y arriver et V pour aller de $j + 1$ et arriver au dépôt final.

Si nous fixons $X = \{s\} \cup \{\cup_{j=0, \dots, M+1} X(j)\}$, alors nous appelons **Useful VD_Subgraph** le sous-graphe de **VD_Graph** induit par X .

Exemple 9 Le sous-graphe **Useful VD_Subgraph** de la figure (6.7) a été construit avec les valeurs $M = 4$, $C^{Veh} = 6$, $E_0 = 3$, $\alpha = \beta = 1$, et les valeurs t , d , e , ε données par le tableau (6.20) (nous ne mentionnons pas ici les valeurs de temps t , d , $TMax$). Les valeurs de X sont :

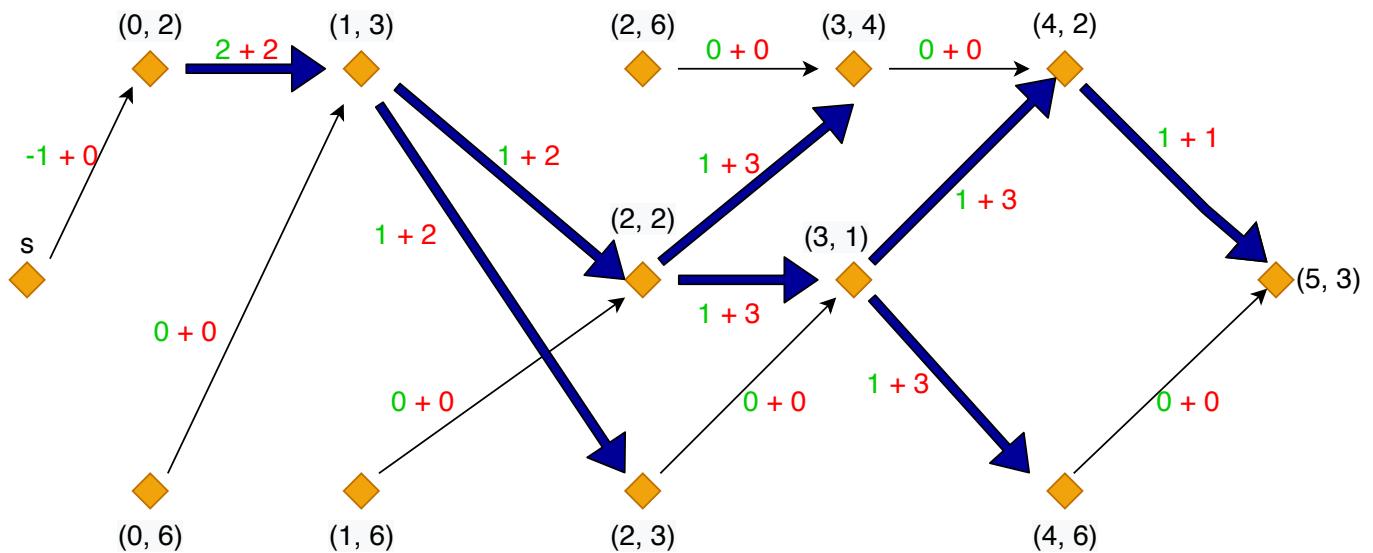
- $\square X(4 + 1) : p = ((4 + 1), 3) = (5, 3)$
- $\square X(4) : \{(4, \varepsilon_4), (4, E_0 + e_4)\} = \{(4, 2), (4, 3 + 3)\} = \{(4, 2), (4, 6)\}$
- $\square X(3) : \{(3, \varepsilon_3), (3, \varepsilon_4 + e_3)\} = \{(3, 1), (3, 2 + 2)\} = \{(3, 1), (3, 4)\}$
 $(3, E_0 + e_4 + e_3)$ est éliminé car $E_0 + e_4 + e_3 > C^{Veh}$
- $\square X(2) : \{(2, \varepsilon_2), (2, \varepsilon_3 + e_2), (2, \varepsilon_4 + e_3 + e_2)\} = \{(2, 2), (2, 1 + 2), (2, 2 + 2 + 2)\} = \{(2, 2), (2, 3), (2, 6)\}$

- $\square X(1) : \{(1, \varepsilon_1), (1, \varepsilon_2 + e_1)\} = \{(1, 3), (1, 2 + 4)\} = \{(1, 3), (1, 6)\}$
 $(1, \varepsilon_3 + e_2 + e_1) = (1, 3 + e_1)$ et $(1, \varepsilon_4 + e_3 + e_2 + e_1) = (1, 6 + e_1)$ sont éliminés car $3 + e_1 > C^{Veh}$
et $6 + e_1 > C^{Veh}$
- $\square X(0) : \{(0, \varepsilon_0), (0, 3 + E_0)\} = \{(0, 2), (0, 3 + 3)\} = \{(0, 2), (0, 6)\}$
 $(0, 6 + E_0)$ est éliminé car $6 + E_0 > C^{Veh}$

j	0	1	2	3	4	5=0
e_j	3	4	2	2	3	*
$\varepsilon_j = \varepsilon_j^*$	2	3	2	1	2	2
t_j	4	5	3	2	6	*
$d_j = d_j^*$	3	3	4	2	3	4

TABLE 6.20 – Entrées du *Useful VD_Subgraph*.

Les flèches épaisses représentent ici les déplacements des véhicules qui inclus un détour de recharge. Les chiffres de couleur rouge représentent les composantes énergétiques des coûts. Les chiffres de couleur verte représentent les composantes temps des coûts.

FIGURE 6.7 – Le sous-graphe **Useful VD_Subgraph**.

La construction de **VD_Graph** et le lemme 3 nous permettent d'affirmer le théorème 6.

Démonstration : En raison du lemme 3 et de la construction du sous-graphie *Useful VD_Subgraph* de la figure (6.7), nous voyons que résoudre **VD** signifie rechercher le plus court chemin de s à p dans le sous-graphie *Useful VD_Subgraph*. Mais, pour tout j , la cardinalité de $X(j)$ ne dépasse pas $M - j + 1$, et le nombre d'arcs qui relie $X(j)$ à $X(j + 1)$ ne dépasse pas $Card(X(j)) + Card(X(j + 1))$.

CHAPITRE 7

ESTIMATION DES COÛTS DE PRODUCTION PAR APPRENTISSAGE

Sommaire

7.1 Introduction	206
7.2 Réseaux de neurones dont les entrées sont des données brutes	209
7.2.1 Construction des réseaux SIMPLE_TYPE et SIMPLE_PERIODE pour la prédiction du coût de production	209
Description des entrées des réseaux de neurones SIMPLE_	210
Description des architectures des réseaux de neurones SIMPLE_	210
Description des sorties des réseaux de neurones SIMPLE_	212
7.2.2 Construction du réseau MIXTE pour la prédiction du coût de production et du numéro de période de la dernière recharge	212
Description des entrées du réseau de neurones MIXTE	212
Description de l'architecture du réseau de neurones MIXTE	212
Description des sorties du réseau de neurones MIXTE	215
7.2.3 Cas des instances aux nombres de périodes hétérogènes	215
7.3 Réseaux de neurones dont les entrées sont des indicateurs	216
7.3.1 Indicateurs	216
7.3.2 Construction du réseau INDIC_TEMPS pour la prédiction du numéro de période de la dernière recharge	219
Description des entrées du réseau de neurones INDIC_TEMPS	219
Description de l'architecture du réseau de neurones INDIC_TEMPS	220
Description des sorties du réseau de neurones INDIC_TEMPS	221
7.3.3 Construction du réseau INDIC_COUT pour la prédiction du coût de production	221
Description des entrées du réseau de neurones INDIC_COUT	221
Description de l'architecture du réseau de neurones INDIC_COUT	222
Description des sorties du réseau de neurones INDIC_COUT	225
7.4 Expérimentations numériques	225
7.4.1 Objectifs et contexte technique	225
7.4.2 Instances	226
Analyse statistique des instances	226
7.4.3 Moyenne de la valeur absolue des gaps	230
7.4.4 Résultats des réseaux SIMPLE_TYPE et SIMPLE_PERIODE	232

Résultats du réseau SIMPLE_TYPE	233
Résultats du réseau SIMPLE_PERIODE	234
7.4.5 Résultats du réseau MIXTE_COUT pour la prédition du coût de production . .	236
7.4.6 Résultats du réseau INDIC_COUT pour la prédition du coût de production . .	237
7.4.7 Résultats du réseau MIXTE_TEMPS pour la prédition du numéro de période de la dernière recharge	239
7.4.8 Résultats du réseau INDIC_TEMPS pour la prédition du numéro de période de la dernière recharge	240
7.5 Conclusion	242

7.1 Introduction

Noms	Définitions
Itérations	Nombre de fois que l'ensemble des données sera passé au réseau de neurones lors de la phase d'entraînement
Batch	Nombre d'échantillons qui passeront par le réseau de neurones à chaque cycle d'apprentissage
Cycle d'apprentissage	Un nouveau cycle d'apprentissage commence à chaque fois que les poids synaptiques du réseau sont recalculés

TABLE 7.1 – Glossaire.

Dans ce chapitre, on cherche à construire un outil d'aide à la décision basé sur les réseaux de neurones qui pourra estimer rapidement le coût de production associé à une stratégie de recharge réduite décrite à la section (6.4.5) du chapitre précédent. Autrement dit, on construit donc ici un estimateur rapide des coûts de production. Cet outil pourrait servir à aider un décideur à prendre des décisions sur des stratégies de recharge réduite afin de limiter les risques de perte (de temps ou d'argent). Par exemple le décideur pourrait avoir plusieurs stratégies de recharge réduite et voudrait décider rapidement quelle stratégie réaliser.

Cet estimateur pourrait aussi par exemple servir à optimiser la tournée du véhicule. Dans les chapitres précédents, on a présenté plusieurs méthodes de résolution du problème SMEPC. On a notamment présenté au chapitre 6 un schéma collaboratif nommé Pipe-line VD_PM car il a été constaté que le problème SMEPC peut être divisé en deux sous-problèmes à savoir le problème *Vehicle-Driver* et le problème de production *Production-Manager*. Il ressort de nos expérimentations numériques que parmi les deux algorithmes de résolution des deux sous-problèmes, celui qui est lourd en temps CPU est l'algorithme de résolution du problème *Production-Manager*. Dans les chapitres précédents, on a fixé la tournée du véhicule, autrement dit on a supposé qu'on connaît dans quel ordre les stations seront visitées car notre objectif était de se focaliser uniquement sur les aspects synchronisation du problème SMEPC. L'outil d'aide à la décision construit dans ce chapitre pourrait être utilisé pour optimiser la tournée du véhicule. Pour évaluer le coût de production, on pourrait bien évidemment utiliser le module PM de Pipe-line VD_PM mais ce module étant lourd en temps CPU, on veut construire un réseau plus rapide qui peut remplacer ce module et évaluer le coût de la partie production PM. Pour cela, on peut utiliser cet outil pour obtenir une approximations de la qualité d'une tournée très rapidement.

Les entrées du modèle SMEPC que nous utiliserons pour construire nos réseaux de neurones pour le problème de production sont présentées au tableau (7.2).

Noms	Significations
C^{Tank}	Capacité de la citerne d'hydrogène
C^{Veh}	Capacité du réservoir d'hydrogène du véhicule
N	Nombre de périodes de production
H_0	Charge initiale de la citerne d'hydrogène
$Cost^F$	Coût d'activation de la micro-usine
Pour $i = 0, \dots, N - 1, R_i$	Rendement de production lié à la période i
Pour $i = 0, \dots, N - 1, Cost_i^V$	Coût de production lié à la période i
Q	Nombre d'opérations de recharge en carburant effectuées par le véhicule
Pour $q = 1, \dots, Q, \mu_q$	Quantité d'hydrogène qui est rechargée lors de la $q^{ième}$ opération de recharge
Pour $q = 1, \dots, Q, [m_q, M_q]$	Fenêtre de temps de la $q^{ième}$ opération de recharge

TABLE 7.2 – Entrées utilisées pour construire nos réseaux de neurones.

Les schémas d'apprentissage que nous avons choisi d'explorer ici sont les réseaux de neurones. Il existe plusieurs types de réseaux de neurones. Ces réseaux de neurones se différencient par la manière dont les informations sont propagées entre les différentes couches de neurones. La variante la plus simple est celle du réseau de neurones dit *feed-forward*, ici, les informations passent directement de la couche d'entrée aux couches cachées puis à la couche de sortie. Dans ce chapitre, on s'intéresse uniquement à ce type de réseau de neurones et la fonction d'agrégation utilisée est la somme pondérée. Mais il existe d'autres types de réseaux de neurones à savoir les réseaux de neurones récurrents et les réseaux de neurones convolutifs. Les réseaux de neurones récurrents sauvegardent les résultats produits par les neurones et nourrissent le réseau à l'aide de ces résultats. Ce mode d'apprentissage est un peu plus complexe. Les réseaux de neurones convolutifs sont quant à eux de plus en plus utilisés dans différents domaines : reconnaissance d'images, traitement naturel du langage. En reconnaissance d'images par exemple, un réseau de neurones convolutifs est constitué de deux parties : une première partie dite convulsive qui est chargée de faire la compression (réduction de la dimension) des images et une deuxième partie classification du réseau qui correspond à un réseau Perceptron Multicouche dont l'acronyme est MLP (*Multi Layers Perceptron*). Un MLP est un réseau à propagation *feed-forward* constitué de plusieurs couches.

Le nombre de données d'entrée et les données de sortie du réseau de neurones sont généralement fixés par la nature du problème. Le nombre de couches cachées et le nombre de neurones par couche n'est pas facile à déterminer, cela dépend principalement de la quantité et de la complexité des données. En général, quand on augmente le nombre de neurones cachés, on gagne de la précision sur les données utilisées en apprentissage mais le réseau perd sur son pouvoir de généralisation pour d'autres données, d'où l'expression sur-apprentissage. Ainsi, plus le nombre de couches et de neurones est élevé, plus le réseau aura besoin d'une grande quantité de données pour être entraîné efficacement. Une architecture qui donne de bons résultats pour une application donnée ne peut être déterminée que d'une façon expérimentale. Pour construire un réseau de neurones, on a besoin de fixer un certain nombre de paramètres. Parmi eux, on peut citer la fonction d'activation, le nombre d'itérations (appelé epochs dans le vocabulaire des réseaux de neurones) et la taille du batch. La fonction d'activation permet de normaliser les sorties de neurones dans un intervalle prédéfini. Par exemple, les sorties de neurones peuvent être échelonnées sur un intervalle $[0,1]$ par la fonction sigmoïde : $f(x) = \frac{1}{1 + e^{-x}}$. Le nombre d'itérations représente le nombre de fois que l'ensemble des données sera passé au réseau

de neurones lors de la phase d'entraînement. Plus le nombre d'itérations est grand, plus le réseau sera entraîné longtemps. La taille du batch représente le nombre d'échantillons qui passeront par le réseau de neurones à chaque cycle d'apprentissage. Un nouveau cycle d'apprentissage commence à chaque fois que les poids synaptiques du réseau sont recalculés.

Au début de la phase d'apprentissage, les données d'apprentissage et de validation sont présentées au réseau avec la valeur de sortie (le coût de production) correspondantes. Les valeurs des poids sont ajustées et affinées continuellement tout au long de la phase d'apprentissage. La correction des poids au cours de l'entraînement ne tient compte que des données d'apprentissage. Au cours de cette phase, les poids du réseau sont corrigés de manière à minimiser l'erreur au carré entre la réponse calculée par le réseau et la réponse attendue. Généralement, l'erreur calculée sur les données d'apprentissage diminue continuellement au cours de l'entraînement. Toutefois, une longue phase d'entraînement diminue la capacité de généralisation du réseau en l'adaptant uniquement aux données d'apprentissage. À cet effet, les données de validation déterminent à quel moment l'apprentissage doit être arrêté. Les données de validation représentent généralement 1/3 des données d'apprentissage. Les données de validation servent uniquement à vérifier le comportement du réseau au cours de l'entraînement face à des données qui lui sont étrangères. Généralement, contrairement à l'erreur calculée sur les données d'apprentissage qui diminue continuellement au cours de l'entraînement, celle calculée sur les données de validation diminue dans la première phase d'entraînement en suivant une allure semblable à celle des données d'apprentissage avant de commencer une lente ascension. Ceci est expliqué par le fait que le réseau commence à perdre son pouvoir de généralisation en adaptant les poids de ses neurones uniquement aux données d'apprentissage. L'entraînement du réseau sera donc arrêté dès que cette erreur commence son ascension. Toutefois, afin d'éviter un arrêt prématuré d'apprentissage causé par une augmentation ponctuelle de l'erreur des données de validation, on introduit souvent un seuil de décision qui tolère des légères ascensions successives de l'erreur. Si l'erreur sur les données de validation continue son ascension au-delà de ce seuil, on arrête l'apprentissage du réseau et on conserve les valeurs des poids qui correspondent à l'itération qui précède cette ascension.

Après l'arrêt de la phase d'apprentissage, on vérifie la performance du réseau avec les données de test. Ces données n'ont pas servi à l'apprentissage et n'ont joué aucun rôle dans la prise de décision dans l'arrêt de l'apprentissage. Les données de test sont utilisées uniquement pour mesurer la performance du réseau après l'arrêt de l'apprentissage. Si le réseau arrive à faire des prédictions correctes avec un gap acceptable, on peut dire que le réseau est opérationnel. Dans le cas contraire, il faut réviser le réseau et recommencer l'apprentissage.

Dans la **section** 7.2, on présente trois réseaux de neurones : **SIMPLE_TYPE**, **SIMPLE_PERIODE** et **MIXTE**. Les réseaux **SIMPLE_TYPE** et **SIMPLE_PERIODE** prédisent le coût de production. Le réseau **MIXTE** contrairement aux réseaux **SIMPLE_TYPE** et **SIMPLE_PERIODE**, essaye d'épouser les caractéristiques du problème. Le réseau **MIXTE** prédit le coût de production (on nomme cette partie du réseau **MIXTE_COUT**) et le numéro de période de la dernière recharge (on nomme cette partie du réseau **MIXTE_TEMPS**). Puisque les données d'entrées des réseaux de neurones **SIMPLE_TYPE**, **SIMPLE_PERIODE** et **MIXTE** sont nombreuses cela a pour effet d'augmenter considérablement le nombre de poids. Dans ce chapitre, on va se focaliser uniquement sur des instances à 20 périodes. Pour pallier à ce problème d'augmentation du nombre de poids, on va construire d'autres réseaux de neurones dont les entrées seront des indicateurs. On va définir un ensemble d'indicateurs qui vont représenter nos données de façon plus compacte. Dans la **section** 7.3, on commence par présenter les indicateurs qui serviront d'entrée aux réseaux de neurones basés sur les indicateurs. On finit cette

section en présentant l'architecture de nos réseaux de neurones basés sur les indicateurs qu'on nomme **INDIC_TEMPS** et **INDIC_COUT** qui font respectivement une approximation du numéro de période de la dernière recharge et du coût de production. Dans la **section 7.4**, on montre les résultats des expérimentations numériques de tous les réseaux de neurones. On présente au préalable les instances qui ont été utilisées pour entraîner les réseaux. Pour simplifier le problème, on ne travaille qu'avec des instances qui ont le même nombre de périodes.

7.2 Réseaux de neurones dont les entrées sont des données brutes

Cette section présente la méthodologie suivie pour la conception et l'apprentissage des réseaux de neurones pour la prédiction du coût de production correspondant à une stratégie de recharge réduite. Plus précisément, on décrit deux réseaux qu'on nomme **SIMPLE_TYPE** et **SIMPLE_PERIODE** et un réseau qu'on nomme **MIXTE** en définissant les éléments suivants : le format des données d'entrées, l'architecture des réseaux de neurones, les fonctions d'activation et l'algorithme d'apprentissage. Les réseaux de neurones de cette section ont été implémenté en python à l'aide de Keras qui dispose de plusieurs modules de création et d'entraînement de réseaux de neurones.

Tout au long de ce chapitre, nous utiliserons les expressions suivantes :

- Layer Dense* correspond à des couches dont les neurones reçoivent en entrée une valeur égale à la somme pondérée de l'ensemble des neurones de la couche précédente ;
- Multiply Dense* permet de multiplier la valeur de sortie de 2 couches de neurones terme à terme.
- Layer Concatenate* recopie les neurones de leur couche initiale vers la couche de concaténation sans modification de leur valeur de sortie

7.2.1 Construction des réseaux SIMPLE_TYPE et SIMPLE_PERIOD pour la prédiction du coût de production

Lorsqu'on parle de réseau **SIMPLE_** on fait allusion aux deux réseaux nommés **SIMPLE_TYPE** et **SIMPLE_PERIOD**. Dans cette section, notre objectif est de construire deux réseaux de neurones « simple » qui serviront de base pour la comparaison avec les réseaux plus sophistiqués. Concernant les entrées du réseau, nous allons tester deux façons de les agencer qu'on nomme ici agencement par période et par type. Autrement dit, on a classé les données par type (le réseau obtenu s'appelle **SIMPLE_TYPE**) et par période (le réseau obtenu s'appelle **SIMPLE_PERIOD**). Par exemple, pour une instance de trois périodes, si les données sont :

- Le coût fixe est $A = [1, 1, 1]$;
- Le coût variable est $Cost_i^V = [5, 5, 5]$;
- La date de début de la dernière recharge est 2;
- Le rendement est $R = [10, 10, 10]$;
- La recharge est $Recharge = [0, 3, 0]$.

Si les données sont classées par **type** alors elles seront agencées de la façon suivante « 1, 1, 1, 5, 5, 5, 2, 10, 10, 10, 0, 3, 0 ». Si les données sont classées par **période** alors elles seront agencées de la façon suivante « 1, 5, 10, 0, 1, 5, 10, 3, 1, 5, 10, 0, 2 ». Concernant les fonctions d'activation, on va tester plusieurs combinaisons pour voir celles qui fonctionnent le mieux pour notre problème.

Dans les réseaux **SIMPLE_TYPE** et **SIMPLE_PERIODE**, les entrées de chaque couche sont les sorties de la couche précédente. Et chaque neurone de chaque couche est connecté à tous les neurones de la couche précédente. Dans cette section, on va décrire les données d'entrée, l'architecture et les données de sortie de notre réseau.

Description des entrées des réseaux de neurones SIMPLE_

Les données utilisées lors de la phase d'entraînement des réseaux de neurones doivent être pertinentes pour la prédiction du coût de production. Dans notre cas, les données d'entrée seront les suivantes :

- Le coût d'activation $A = A_i$, avec $i \in \{0, \dots, N - 1\}$ puisque dans notre problème le coût d'activation est $Cost^F$, on pose ici $\forall i, A_i = Cost^F$, N est le nombre de périodes ;
- Pour $i = 0, \dots, N - 1$, $Cost_i^V$ est la coût de production lié à la période i ;
- Le numéro de période du début de la dernière recharge
- Pour $i = 0, \dots, N - 1$, R_i est le rendement de production lié à la période i ;
- Q est le nombre d'opérations de recharge en carburant effectuées par le véhicule ;
- μ_q est la quantité d'hydrogène qui est rechargée lors de la $q^{ième}$ opération de recharge ;
- Des bornes inférieures m_1, \dots, m_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en hydrogène ont lieu ;
- Un vecteur $Recharge$ rempli de la façon suivante : Pour $q = 1, \dots, Q$, μ_q et $i \in [m_q, M_q]$, $Recharge_i = \mu_q$ sinon $Recharge_i = 0$.

Description des architectures des réseaux de neurones SIMPLE_

Ces réseaux de neurones sont complètement connecté c'est-à-dire que tous les neurones d'une couche sont connectés à tous les neurones de la couche précédente. Dans notre réseau, le nombre de couches et le nombre de neurones des couches a été choisi assez faible pour ne pas avoir trop de poids. L'architecture de ces réseaux de neurones que nous avons conçu est la suivante :

- Elle a une couche d'entrées constituée de huit neurones
- Elle a une couche cachée constituée de quatre neurones
- Elle a une couche de sortie constituée d'un neurone

La figure (7.1) représente l'architecture du réseau **SIMPLE_TYPE**, ici les données sont classées par type. La figure (7.2) représente l'architecture du réseau **SIMPLE_PERIODE**, ici les données sont classées par période.

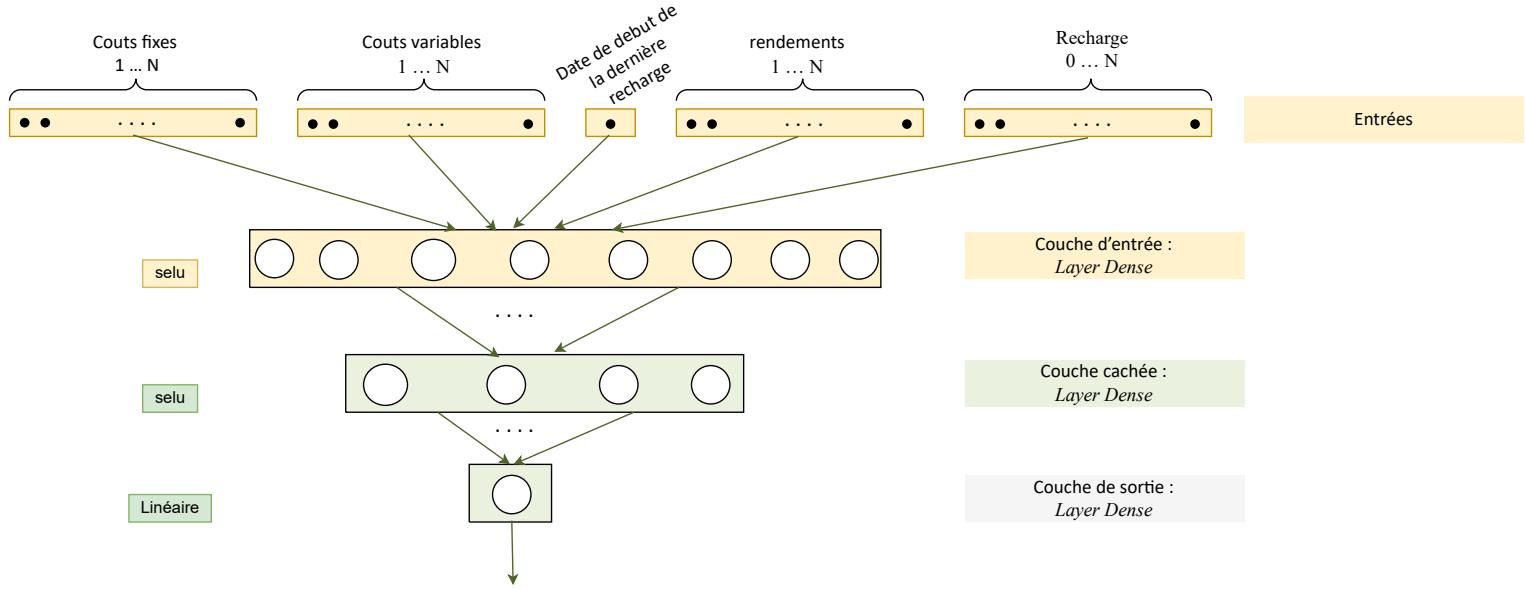


FIGURE 7.1 – Réseau SIMPLE_TYPE prédisant le coût de production.

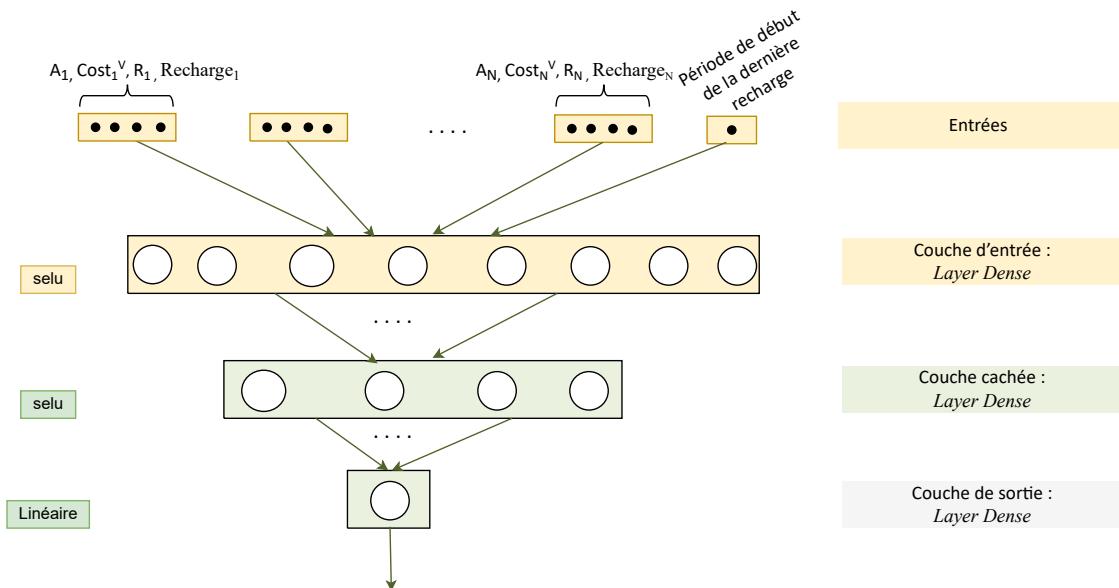


FIGURE 7.2 – Réseau SIMPLE_PERIODE prédisant le coût de production.

Le nombre de poids et de biais des réseaux de neurones SIMPLE_TYPE et SIMPLE_PERIODE dépend de la taille des instances, on a : $8 \times (4N + 2)$ poids à la couche d'entrée et 8 biais. Où N est le nombre de périodes. On a $4 \times 8 = 32$ poids et 4 biais à la couche cachée. On a 4 poids à la couche de sortie et 1 biais. Dans ce chapitre, puisqu'on travaille uniquement avec des instances de 20 périodes, la couche d'entrées a $8 \times (4 \times 20 + 2) = 656$ poids. Si on veut traiter des instances dont le nombre de périodes est hétérogène, on peut par exemple compresser ces instances. L'algorithme d'optimisation utilisé est la descente de gradient stochastique.

Les données de validation sont 1/3 des données d'apprentissage. Le nombre d'itérations correspondant à l'itération qui a fourni le plus petit gap sur les données de test est 20 pour le réseau SIMPLE_TYPE et 3 pour le réseau SIMPLE_PERIODE. La taille des batchs fixée ici est 32.

Description des sorties des réseaux de neurones SIMPLE_

La sortie des réseaux de neurones **SIMPLE_TYPE** et **SIMPLE_PERIODE** est le coût de production.

Dans la section suivante, on va construire le réseau **MIXTE** qui épouse la logique de notre problème.

7.2.2 Construction du réseau MIXTE pour la prédiction du coût de production et du numéro de période de la dernière recharge

Notre objectif ici est de construire un réseau de neurones qu'on nomme **MIXTE** qui épouse les mécanismes de calcul et les caractéristiques du problème de production. Dans cette section, on va décrire les données d'entrée, l'architecture et les données de sortie de notre réseau.

Description des entrées du réseau de neurones MIXTE

Les entrées du réseau de neurones **MIXTE** sont :

- Le coût d'activation $A = A_i$, avec $i \in \{0, \dots, N - 1\}$ puisque dans notre problème le coût d'activation est $Cost^F$, on pose ici $\forall i, A_i = Cost^F$;
- Pour $i = 0, \dots, N - 1$, $Cost_i^V$ est le coût de production lié à la période i ;
- Pour $i = 0, \dots, N - 1$, R_i est la rendement de production lié à la période i ;
- Q est le nombre d'opérations de recharge en carburant effectuées par le véhicule ;
- μ_q est la quantité d'hydrogène qui est rechargée lors de la $q^{ième}$ opération de recharge ;
- Des bornes inférieures m_1, \dots, m_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en hydrogène ont lieu ;
- On a N périodes $1, \dots, N$ et 2 périodes fictives $0, N + 1$; Q recharges +2 recharges fictives $(0, Q + 1)$ qui modélisent les niveaux de départ et d'arrivée de la citerne ; μ_1, \dots, μ_Q : quantité à recharger par le véhicule ; F_1, \dots, F_Q fenêtres de temps pour les recharges ($F_q = [m_q, M_q]$) ; L_1, \dots, L_Q longueurs des fenêtres (NB. on suppose connaître exactement les périodes de recharge donc $L_i = 1 \ \forall i = 1, \dots, N$) ; $\forall q = 1, \dots, Q$, $\mu_q^* = \mu_q / L_q$ (NB. actuellement on aura $\mu_q^* = \mu_q$)
 $\forall i \in 1, \dots, N, \lambda_i = \sum_{q=1, \dots, Q \vee i \in F_q} \mu_q^*$ (permet de se ramener à un vecteur indicé sur les périodes)
- $\lambda_0 = -H_0$ et $\lambda_{N+1} = H_0$

Exemple 10 Calcul du λ : $N = 6, Q = 2$ (5 périodes et 2 recharges)

$$F_1 = [2, 3], F_2 = [3, 6], \mu_1 = 8, \mu_2 = 20, L_1 = 2, L_2 = 4, H_0 = 3$$

$$\mu_1^* = 4, \mu_2^* = 5$$

$$\lambda_0 = -3, \lambda_7 = 3 \text{ (quantités liées aux périodes fictives)} \quad \lambda_1 = 0, \lambda_2 = \mu_1^* = 4, \lambda_3 = \mu_1^* + \mu_2^* = 9, \\ \lambda_4 = \mu_2^* = 5, \lambda_5 = \mu_2^* = 5, \lambda_6 = \mu_2^* = 5$$

Description de l'architecture du réseau de neurones MIXTE

Notre objectif est de représenter avec un réseau de neurone la formule de calcul du coût de production. On divise nos données en deux catégories à savoir les prix ($Cost_i^V, Cost^F$) et les valeurs qui désignent les quantités d'hydrogène (R_i, λ_i). Puis, on normalise chaque catégorie avec la fonction sigmoïde. On sait que le coût de production est constitué de deux composantes à savoir le coût

d'activation de la micro-usine (coût fixe) et le coût de production variable. On sait que les vecteurs désignant les périodes de production et les périodes de démarrage de la micro-usine sont des vecteurs de booléens. Pour chaque période i , on veut connaître la probabilité γ_i qu'elle soit une période de production et la probabilité γ_i^* qu'elle soit une période de démarrage de la micro-usine. Une fois qu'on connaît les probabilités γ_i et γ_i^* , il suffira de les multiplier (à l'aide d'une couche *Multiply Dense*) respectivement par le coût variable $Cost_i^V$ et par le coût fixe $Cost^F$ puis de sommer les valeurs de toutes les périodes pour obtenir le coût de production car la formule de calcul du coût de production est $COST = \sum_{i=0, \dots, N-1} \gamma_i \times Cost_i^V + \gamma_i^* \times Cost^F$. La probabilité γ_i est calculée avec la fonction sigmoïde. La probabilité γ_i^* se déduit du vecteur des γ_i car si on connaît les périodes de production, on peut aisément déduire quelles sont les périodes d'activation de la micro-usine. Pour le calcul de la probabilité τ_i que i soit la dernière période de recharge on utilise la fonction softmax car on a qu'une unique dernière période de recharge donc il faut que $\sum_{i=0, \dots, N-1} \tau_i = 1$. On veut éviter que le calcul du numéro de période de la dernière recharge $T = \sum_{i=1, \dots, N} i \times \tau_i$ soit faussé. L'architecture du réseau de neurones (Voir figure (7.3)) que nous avons conçu est la suivante :

- Elle a deux couches d'entrées constituées chacune de N neurones ;
- Elle a quatre couches cachées constituées chacune de N neurones ;
- Elle a deux couches de sortie constituée l'une de N neurones et l'autre d'un neurone.

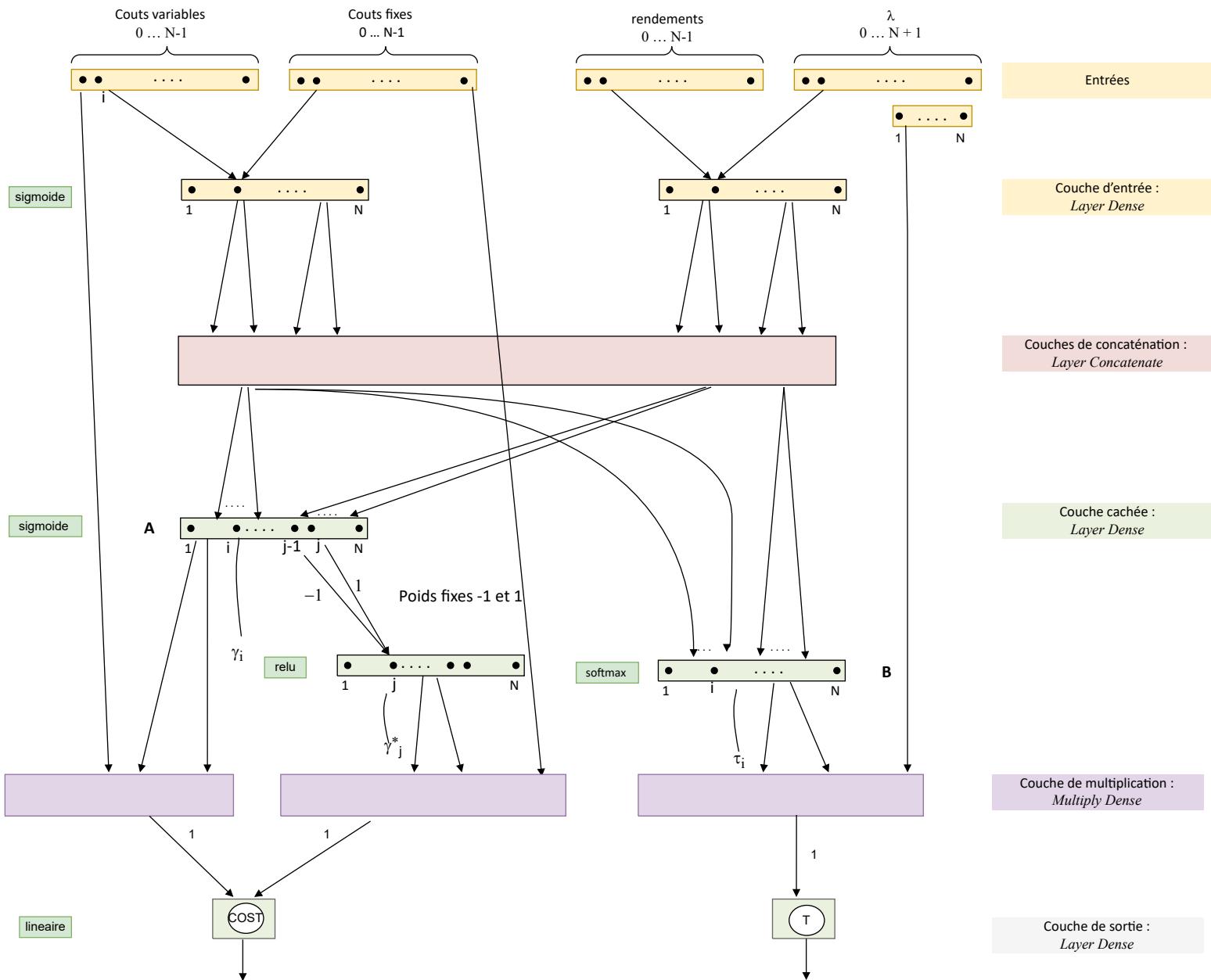


FIGURE 7.3 – Réseau MIXTE prédisant le coût de production et le numéro de période de la dernière recharge.

Les neurones du réseau ici ne sont pas complètement connectés c'est-à-dire que les neurones d'une couche sont connectés à certains neurones de la couche précédente. Le nombre de neurones des couches a été choisi en fonction du nombre de périodes. Les fonctions d'activation utilisées sont la fonction sigmoïde, la fonction softmax, la fonction RELU (Elle vaut x si x est supérieur à 0, et 0 sinon. Autrement dit, c'est le maximum entre x et 0) et la fonction linéaire.

Le nombre de poids et de biais du réseau de neurones MIXTE est **quadratique** en fonction de N et de la taille des instances. Le nombre de poids est : $2 \times N \times N + (2N + 2) \times N + 2 \times N \times N = 6N^2 + 2N$. Le nombre de biais est $N + N + N$. Ce qui fait 2500 poids et biais en tout avec $N = 20$. L'algorithme d'optimisation utilisé est la descente de gradient stochastique. Les données de validation sont 1/3 des données d'apprentissage. La taille des batchs fixée ici est 32.

Pour diminuer le nombre de poids du réseau MIXTE on décide de supprimer certains poids de la façon suivante : chaque neurone i d'une couche est connecté uniquement aux $i - 2, i - 1, i,$

$i + 1, i + 2$ de la couche précédente. Lorsqu'on sélectionne les voisins le nombre de neurones devient $2 \times N \times N + (2N + 2) \times N + 2 \times N \times N = 188 + 194 + 188 = 570$.

La valeur V est calculée par la formule $V = \alpha(\sum_{i=1,\dots,N} i \times \tau_i) + \beta(\sum_{i=1,\dots,N} Cost_i^V \times \gamma_i + \sum_{i=1,\dots,N} A_i \times \gamma_i^*)$. Ainsi le numéro de période de la dernière recharge est $\sum_{i=1,\dots,N} i \times \tau_i$ et le coût de production est $\sum_{i=1,\dots,N} Cost_i^V \times \gamma_i + \sum_{i=1,\dots,N} A_i \times \gamma_i^*$.

Lorsqu'on ne veut traiter que la partie production avec le réseau de neurones de la figure (7.3) on supprime de la couche B à la couche de sortie du réseau. On nomme le réseau ainsi obtenu **MIXTE_COUT**. Lorsqu'on ne veut traiter que la partie recharge avec le réseau de neurones de la figure (7.3) on supprime de la couche A à la couche de sortie du réseau. On nomme le réseau ainsi obtenu **MIXTE_TEMPS**. Le nombre d'itérations correspondant à l'itération qui a fourni le plus petit gap sur les données de test est 161 pour le réseau **MIXTE_COUT** et 195 pour le réseau **MIXTE_TEMPS**.

Description des sorties du réseau de neurones MIXTE

La sortie de ce réseau de neurones est un vecteur de taille $3N$, leur signification est :

- $\gamma_i \in [0, 1], i = 1, \dots, N$: la probabilité que i soit une période de production
- $\gamma_i^* \in [0, 1], i = 1, \dots, N$: la probabilité que i soit une période de démarrage de la micro-usine
- $\tau_i^* \in [0, 1], i = 1, \dots, N, \sum_i \tau_i = 1$ la probabilité que la période i soit la période de dernière recharge

γ_i et γ_i^* sont agrégés pour calculer le coût de production.

7.2.3 Cas des instances aux nombres de périodes hétérogènes

Les réseaux **SIMPLE_TYPE**, **SIMPLE_PERIODE**, **MIXTE_COUT** et **MIXTE_TEMPS** prennent en entrées des données dont la taille doit être fixe, cette taille dépend du nombre de périodes. Or, comme on peut avoir des instances dont le nombre de périodes est différent (on appelle cela des instances hétérogènes). Pour utiliser ces réseaux sur de telles instances, il va falloir avant tout effectuer un pré-traitement qui consistera à rendre ces instances homogènes, c'est-à-dire faire qu'elle ait le même nombre de périodes.

Si on décide de considérer que la taille des données d'entrées est la taille de l'instance dont le nombre de période est le plus élevé alors la matrice des entrées sera une matrice creuse car les entrées sont de tailles différentes. On peut la remplir avec des zéros ou alors la remplir de façon cyclique en répétant les données.

Par contre, si on décide de ramener toutes les instances à une taille unique et si par exemple cette taille unique est 20, on peut effectuer la transformation suivante :

- Si N se situe entre $20.K$ et $20.(K + 1)$ avec $K \geq 1$, alors, pour tout u allant de 1 à $[N/K]$, on fusionne les coefficients des différents vecteurs indexés de 1 à N de la façon suivante :
 - S'il s'agit de R_i qui donne les rendements, on pose $R_u^* = [(\sum_{k,u \leq i < (k+1).u} R_i)/K]$;
 - S'il s'agit de $Cost_i^V$ qui donne les coûts de production, on pose $Cost^V * u = [(\sum_{k,u \leq i < (k+1).u} Cost_i^V)/K]$;
 - S'il s'agit du coût d'activation $Cost^F$, on pose $Cost_u^F = [Cost^F/K]$;
 - Pour ce qui est des fenêtres de temps $[m_q, M_q]$ et de μ_q , $q = 1, \dots, Q$, sur les périodes et quantités de recharge, on divise chaque quantité m_q , M_q et μ_q par K (division entière).

- Quand on obtient le résultat $Cost$ et T , où $Cost$ est le coût et T le numéro de période de la dernière recharge, alors on multiplie $Cost$ et T par K pour reconstituer le résultat souhaité.

Dans la section suivante, on va construire un autre réseau de neurones dont les entrées sont des indicateurs.

7.3 Réseaux de neurones dont les entrées sont des indicateurs

Dans les réseaux de neurones précédent, le nombre de poids et de biais du réseau dépend fortement de la taille des instances. Donc si on a des entrées de grande taille on aura un grand nombre de poids et de biais. Pour éviter que la taille de nos réseaux de neurones dépende de la taille des instances on définit dans cette section un ensemble d'indicateurs. Dans cette section, on compacte les entrées de nos réseaux de neurones en un ensemble d'indicateurs de telle sorte que le réseau ait une taille fixe. Autrement dit, on essaye d'agrégner les entrées. Au vu de la manière dont les indicateurs évoluent, on suppose qu'on peut les utiliser pour obtenir une approximation du coût de production et du numéro de période de la dernière recharge. Pour cela, on va construire des équations dont les coefficients seront trouvées par des réseaux de neurones. On présente ici les réseaux de neurones dont les entrées sont des indicateurs. Ces réseaux seront utilisés pour apprendre la valeur du coût de production et du numéro de période de la dernière recharge. Les entrées du modèle SMEPC que nous utiliserons pour construire nos réseaux de neurones pour le problème de production sont présentées au tableau (7.2). En plus de ces données, nous aurons aussi besoin des données suivantes :

- Le coût d'activation $A = A_i$, avec $i \in \{0, \dots, N - 1\}$ puisque dans notre problème le coût d'activation est $Cost^F$, on pose ici $\forall i, A_i = Cost^F$;
- Q est le nombre d'opérations de recharge en carburant effectuées par le véhicule;
- μ_q est la quantité d'hydrogène qui est rechargée lors de la $q^{ième}$ opération de recharge;
- Des bornes inférieures m_1, \dots, m_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en hydrogène ont lieu;
- Des bornes supérieures M_1, \dots, M_Q respectivement pour les numéros de période $i_1, \dots, i_Q \in \{0, \dots, N - 1\}$ lorsque les opérations de recharge en hydrogène ont lieu;
- Le coût unitaire de production est $P_i = Cost_i^V / R_i$
- L'écart du coût unitaire de production est $G_i = |P_{i+1} - P_i|$. G_i est une mesure de désordre des coûts unitaires par rapport au temps. Cette valeur sanctionne le fait que les coûts de production oscillent beaucoup lorsqu'on passe de i à $i + 1$.

7.3.1 Indicateurs

Dans cette section, on définit des indicateurs qui permettent de faire une approximation du coût de production et du temps de parcours du véhicule. On va définir des indicateurs basés sur les moyennes de nos données. On veut calculer des quantités agrégées qui auront un impact prévisible sur le coût de production. Ces quantités agrégées seront utilisées comme entrées de nos réseaux de neurones afin que la taille de ceux-ci ne dépende plus de la taille des instances. On veut donc avoir des réseaux de taille fixe. Les indicateurs encadrés sont ceux qui sont utilisés comme entrées sans aucune modification. Nous introduisons plusieurs indicateurs comme suit :

□ Indicateurs énergétiques :

- $Mu = \sum_q \mu_q$: Mu est la quantité totale d'hydrogène rechargée. Si la valeur de Mu augmente alors la valeur du coût de production a tendance à aussi augmenter ;
- $K = \lceil Mu/C^{Tank} \rceil$: K représente le nombre minimum de fois qu'il faudra faire le plein de la citerne de la micro-usine pour pouvoir satisfaire toutes les recharges. K représente aussi le nombre minimum de démarrage de la machine de production. Si la valeur de K augmente alors la valeur du coût de production a tendance à aussi augmenter ;
- $R = (\sum_i R_i)/N$: R est la quantité moyenne de carburant qu'on peut produire par période. Si la valeur de R augmente alors la valeur du coût de production a tendance à diminuer car on aura besoin de moins de périodes pour produire le carburant dont on a besoin ;
- $R^0 = (\sum_i |R_i - R|)/N$: R^0 est l'écart type des rendements. Cette valeur mesure le désordre dans le temps des rendements de production, plus cette valeur est proche de zéro, plus les R_i sont proches de R .

□ Indicateurs libres :

- $C = (\sum_i R_i)/Mu$: C représente la difficulté qu'on a à produire le carburant dont le véhicule aura besoin lors de ses recharges. Plus la valeur C est élevée, plus on aura tendance à pouvoir facilement produire du carburant ;
- $C(q) = (\sum_{i \leq M_q} R_i)/(\mu_1 + \dots + \mu_q - H_0)$: $C(q)$ représente la difficulté qu'on a à produire le carburant dont le véhicule aura besoin à la recharge q . Plus les valeurs $C(q)$ sont élevées, plus on aura tendance à pouvoir facilement produire du carburant pour satisfaire les recharges 1 à q ;
- $C^* = \inf_q C(q)$: plus la valeur de C^* augmente , plus on aura tendance à pouvoir facilement produire du carburant dont aura besoin le véhicule. Donc plus cette valeur est grande, plus le numéro de période de la dernière recharge aura tendance à diminuer ;

□ Indicateurs de temps :

- I^0 est la plus petite valeur de l'indice i_0 tel que $(\sum_{i \leq i_0} R_i) \geq Mu$; si on décidait de produire tout le carburant à recharger Mu durant des périodes consécutives en commençant à la première période, I^0 est le nombre de périodes qui seraient nécessaire pour produire tout ce que le véhicule va recharger. $I^0/N \in]0, 1]$: plus cette valeur se rapproche de 1, plus on aura tendance à avoir besoin de périodes consécutives pour satisfaire les recharges, ce qui repoussera le numéro de période de la dernière recharge.
- $R^* = (\sum_i i \times R_i)/(\sum_i R_i)$: R^* est la moyenne des rendements pondérés par les numéros de périodes. Plus cette valeur est grande, plus les meilleurs rendements de production (les plus élevés) auront tendance à se trouver vers la fin de l'espace temps (proche de la période N) ;
- $P^* = (\sum_i i \times P_i)/(\sum_i P_i)$: P^* est la moyenne des coûts unitaires pondérés par les numéros de périodes. Plus cette valeur est grande, plus les plus mauvais coûts unitaires de production (les plus élevés) auront tendance à se trouver vers la fin de l'espace temps (proche de la période N) ;
- $V^* = (\sum_i i \times Cost_i^V)/(\sum_i Cost_i^V)$: V^* est la moyenne des coûts variables pondérés par les numéros de périodes. Plus cette valeur est grande, plus les plus mauvais coûts variables

de production (les plus élevés) auront tendance à se trouver vers la fin de l'espace temps (proche de la période N) ;

- $A^* = (\sum_i i \times A_i)/(\sum_i A_i)$: A^* est la moyenne des coûts fixes pondérés par les numéros de périodes. Plus la valeur A^*/N est grande, plus les plus mauvais coûts fixes de production (les plus élevés) auront tendance à se trouver vers la fin de l'espace temps (proche de la période N) ;
- $G^* = (\sum_i i \times G_i)/(\sum_i G_i)$: G^* permet de localiser le désordre dans le temps des coûts unitaires. Si G^* est petit alors le désordre se trouve au début de l'espace temps (car G_i est grand au début de l'espace temps et petit à la fin de l'espace temps). Si on veut tirer parti de ces oscillations, on doit produire au début de l'espace temps. Si le coût d'activation est faible alors la dernière recharge a tendance à être poussé vers la gauche. Si le coût d'activation est faible et que G^* est grand (forte fluctuation des coûts unitaires vers la droite) alors la date de dernière recharge aura tendance à être poussé vers la droite (vers la fin de l'espace temps) car on cherche à produire au coût le plus petit ;
- $I(q) =$ le plus petit i tel que $R_1 + \dots + R_i \geq \mu_1 + \dots + \mu_q - H_0$; $I(q)$ permet de savoir à partir de quelle période on peut satisfaire la demande μ_q . Autrement dit, la recharge q n'aura pas lieu avant la période $I(q)$. Plus $I(q)$ est grand, plus la date de dernière recharge aura tendance à être poussé vers la droite (vers la fin de l'espace temps) ;
- $\Delta(q) = \text{Sup}(I(q), m_q) - m_q$: $\Delta(q)$ est la correction de la fenêtre de temps de la borne inférieure m_q ;
- $De = \text{Sup}_q \Delta(q)$: De est la plus grande correction de la fenêtre de temps de la borne inférieure m_q . Plus De est grand, plus la date dernière recharge a tendance à être poussé vers la droite (vers la fin de l'espace temps).

□ Indicateurs de coût unitaire :

- $P = (\sum_i P_i)/N$: P est la moyenne des coûts unitaires de production ; plus P augmente, plus le coût de production va avoir tendance à augmenter ;
- $P^0 = (\sum_i |P_i - P|)/N$: P^0 est l'écart type du coût unitaire de production ; P^0 mesure le désordre dans les coûts unitaires de production. Si les coûts unitaires sont élevés et qu'il y a une forte variance dans ceux-ci alors on pourra produire facilement à faible coûts. Si P^0 augmente alors on peut espérer produire sensiblement moins chère que le prix moyen et si c'est le cas le nombre d'activation aura tendance à augmenter.
- $G = (\sum_i G_i)/N$: G est la moyenne des écarts des coûts unitaires de production ; Plus G est grand, plus les coûts de production oscillent beaucoup c'est-à-dire qu'on peut trouver des périodes avec de très petits coûts et des périodes avec de très grands coûts.
- $G^0 = (\sum_i |G_i - G|)/N$: G^0 est l'écart type des écarts des coûts unitaires de production ; G^0 mesure le désordre dans le temps des coûts unitaires de production.

□ Indicateurs de coût absolu :

- $A = (\sum_i A_i)/N$: A est le coût fixe moyen de production de carburant par période ; plus A augmente, plus ça aura tendance à coûter cher de démarrer la micro-usine pour produire du carburant donc le coût de production va avoir tendance à augmenter ;

- $A^0 = (\sum_i |A_i - A|)/N$: A^0 est l'écart type des coûts fixes de production ; cette valeur mesure le désordre dans le temps des coûts d'activation. plus cette valeur est proche de zéro, plus les A_i sont proches de A .
- $V = (\sum_i Cost_i^V)/N$: V est le coût moyen de production variable de carburant par période ; plus V augmente, plus le coût de production va avoir tendance à augmenter ;
- $V^0 = (\sum_i |Cost_i^V - V|)/N$: V^0 est l'écart type des coûts de production variables ; cette valeur mesure le désordre dans le temps des coûts de production variable. plus cette valeur est proche de zéro, plus les V_i sont proches de V . Plus V^0 est grand plus on peut espérer trouver de bonnes valeurs c'est-à-dire qu'on peut espérer produire sensiblement moins chère que le coût de production variable moyen.

Ici, on a raisonné avec la possibilité qu'on a des coûts d'activation variable, mais dans notre problème on a un cout d'activation fixe donc A est le coût d'activation et A^0 vaut 0.

Exemple 11 Reprenons l'exemple (1) avec les entrées $H_0 = 4$, $C^{Tank} = 15$, les valeurs des indicateurs sont $Mu = 25$, $K = 2$, $R = 2,666$, $R^0 = 1,2$, $C = 1,6$, $C(1) = 1,8$, $C(2) = 1$, $C^* = 1$, $I^0 = 8$, $R^* = 7,075$, $P^* = 9,396$, $V^* = 8,385$, $A^* = 8$, $G^* = 4,467$, $I(1) = 1$, $I(2) = 7$, $\Delta(1) = 0$, $\Delta(2) = 0$, $De = 0$, $P = 0,851$, $P^0 = 0,472$, $G = 0,033$, $G^0 = 0,387$, $A = 3$, $A^0 = 0$, $V = 1,733$, $V^0 = 0,684$.

Exemple 12 Reprenons l'exemple (3) avec les entrées $H_0 = 1$, $C^{Tank} = 105$, les valeurs des indicateurs sont $Mu = 96$, $K = 1$, $R = 15,367$, $R^0 = 6,682$, $C = 4,802$, $C(1) = 4,72$, $C(2) = 6,545$, $C(3) = 3,955$, $C(4) = 3,926$, $C^* = 3,926$, $I^0 = 6$, $R^* = 14,575$, $P^* = 13,169$, $V^* = 10,719$, $A^* = 15,5$, $G^* = -0,319$, $I(1) = 2$, $I(2) = 2$, $I(3) = 4$, $I(4) = 6$, $\Delta(1) = 0$, $\Delta(2) = 0$, $\Delta(3) = 0$, $\Delta(4) = 0$, $De = 1$, $P = 0,156$, $P^0 = 0,104$, $G = -0,032$, $G^0 = 0,133$, $A = 13$, $A^0 = 0$, $V = 2,133$, $V^0 = 1,457$.

7.3.2 Construction du réseau INDIC_TEMPS pour la prédiction du numéro de période de la dernière recharge

Dans cette section, on va décrire les données d'entrée, l'architecture et les données de sortie de notre réseau pour la prédiction de la date de la dernière recharge.

Description des entrées du réseau de neurones INDIC_TEMPS

Dans cette section, on identifie les valeurs qui ont une influence pour la prédiction du numéro de période de la dernière recharge. Ces valeurs seront les entrées du réseau de neurones. Les entrées du réseau de neurones INDIC_TEMPS sont :

- H_0/Mu : plus la valeur H_0/Mu est grande, plus la date de la dernière recharge aura tendance à diminuer car on aura tendance à ne pas produire car la quantité d'hydrogène initiale de la citerne est suffisante pour assurer les recharges du véhicule ;
- P^*/N : plus la valeur P^*/N est grande, plus la date de la dernière recharge aura tendance à diminuer car on aura tendance à vouloir produire plus tôt car les meilleurs coûts de production variable s'y trouvent ;
- A^*/N : plus la valeur A^*/N est grande, plus la date de la dernière recharge aura tendance à diminuer car on aura tendance à vouloir produire plus tôt car les meilleurs coûts d'activation s'y trouvent ;

- C : plus la valeur C est grande, plus la date de la dernière recharge aura tendance à diminuer ;
- C^* : plus la valeur C^* est grande, plus la date de la dernière recharge aura tendance à diminuer ;
- K : si la valeur de K augmente alors la valeur de la date de la dernière recharge aura tendance à augmenter ;
- $I^0/N \in]0, 1]$: plus la valeur I^0/N se rapproche de 1, plus le nombre de périodes consécutives dont on aura besoin pour satisfaire les recharges aura tendance à augmenter, ce qui aura tendance à augmenter la date de la dernière recharge ;
- $[m_Q, M_Q]$: c'est l'intervalle de temps durant laquelle la dernière recharge doit être réalisée.

Description de l'architecture du réseau de neurones INDIC_TEMPS

Notre but est de calculer des indicateurs qu'on manipulera à travers des équations pour faire une approximation du numéro de période T de la dernière opération de recharge en carburant. Le calcul de T dépend de l'intervalle $[m_Q, M_Q]$. On cherche un barycentre qui correspond au numéro de période de la dernière recharge compris entre m_Q et M_Q . Si Ga se rapproche de 1 alors la date de la dernière recharge a tendance à se rapprocher de m_Q , il faut ajouter à m_Q la plus grande correction De pour que le véhicule ait le temps de réaliser toutes les recharges. Si Ga se rapproche de 0 alors la date de la dernière recharge a tendance à se rapprocher de M_Q . On va écrire ces équations en respectant le sens des indicateurs, ces derniers seront positifs ou négatifs en fonction de l'impact qu'ils ont sur T sachant que notre objectif global est que T soit le plus petit possible c'est-à-dire qu'il se rapproche de m_Q . Les indicateurs utilisés pour calculer X sont positifs lorsqu'ils poussent X vers la droite (X devient de plus en plus grand). Les indicateurs utilisés pour calculer X sont négatifs lorsqu'ils poussent X vers la gauche (X devient de plus en plus petit). Plus les valeurs positives de X augmentent plus la date de dernière recharge diminue (se rapproche de m_Q). Et, plus les valeurs négatives de X augmentent plus la date de dernière recharge augmente (se rapproche de M_Q). Les coefficients des équations seront trouvés à l'aide d'un réseau de neurones. Donc nous définissons :

- $T = Ga \times (m_Q + De) + (1 - Ga) \times M_Q$ avec $Ga \in [0, 1]$;
- $Ga = \Phi^x(X)$, où Φ^x est une fonction sigmoïde $Exp(x.X)/(Exp(x.X) + 1)$, $x \geq 0$;
- $X = w_0.(H_0/Mu) + w_1.(P^*/N) + w_2.(A^*/N) + w_3.C + w_4.C^* - w_5.K - w_6.I^0/N - w_7$, où $w_0, w_1, \dots, w_6 \geq 0$ et w_7 est non signé.

A partir de ces équations, nous construisons l'architecture du réseau de neurones (Voir figure (7.4)). L'architecture est la suivante :

- Elle a huit couches d'entrées constituées chacune d'un neurone ;
- Elle a six couches cachées ;
- Elle a une couche de sortie d'un neurone.

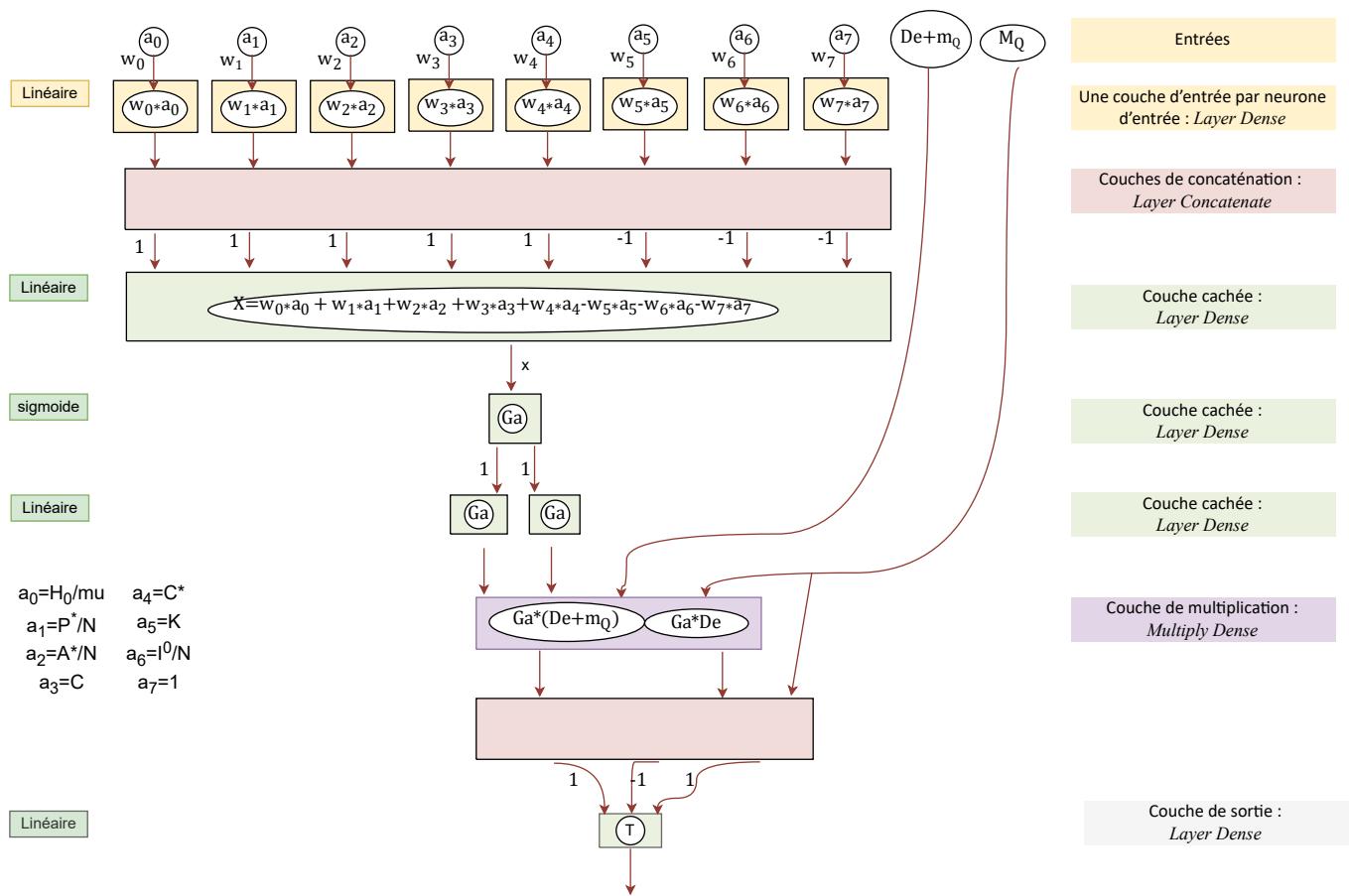


FIGURE 7.4 – Réseau INDIC_TEMPS prédisant le numéro de période de la dernière opération de recharge. Les w_i sont les coefficients synaptiques ou poids.

Les fonctions d'activation utilisées sont la fonction sigmoïde et la fonction linéaire. Le nombre de poids et de biais du réseau de neurones **INDIC_TEMPS** ne dépend pas de la taille des instances. L'algorithme d'optimisation utilisé est la descente de gradient stochastique. Les données de validation sont 1/3 des données d'apprentissage. Le nombre d'itérations correspondant à l'itération qui a fourni le plus petit gap sur les données de test est 3. La taille des batchs fixée ici est 32.

Dans le réseau **INDIC_TEMPS**, on a 9 poids à apprendre.

Description des sorties du réseau de neurones INDIC_TEMPS

La sortie du réseau de neurones **INDIC_TEMPS** est T le numéro de période de la dernière recharge.

7.3.3 Construction du réseau INDIC_COUT pour la prédiction du coût de production

Dans cette section, on va décrire les données d'entrée, l'architecture et les données de sortie de notre réseau pour la prédiction du coût de production.

Description des entrées du réseau de neurones INDIC_COUT

Dans cette section, on identifie les valeurs qui ont une influence pour la prédiction du coût de production. Ces valeurs seront les entrées du réseau de neurones. Les entrées du réseau de neurones **INDIC_COUT** sont :

- P/A : Si cette valeur augmente alors le nombre d'activation aura tendance à aussi augmenter ;
- P^0/A : Si cette valeur augmente alors le nombre d'activation aura tendance à aussi augmenter ;
- G/A : Si cette valeur augmente alors le nombre d'activation aura tendance à aussi augmenter ;
- C/A : Si cette valeur augmente alors le nombre d'activation aura tendance à aussi diminuer ;
- P^*/N : Si P^*/N augmente cela veut dire que les coûts de production sont bons au début alors on aura tendance à produire plus tôt. Donc la date de dernière recharge aura tendance à diminuer et le coût à être plus petit ;
- P^0/P : c'est le désordre dans les prix unitaires de production. si P^0/P augmente cela veut dire qu'il y a du désordre alors on peut produire aux meilleures périodes si le coût d'activation est faible ;
- A/PR : Si A/PR augmente alors le nombre d'activation aura tendance à aussi diminuer ;
- $V0/V$: c'est le désordre dans les coûts de production. c'est l'écart type des coûts de production variable divisé par le coût de production variable moyen ; si V^0/V augmente cela veut dire qu'il y a du désordre alors on peut produire aux meilleures périodes si le coût d'activation est faible ;
- C : si C augmente alors le coût de production aura tendance à aussi diminuer ;
- S/N : S est le nombre de recharge. Si cette valeur augmente alors le nombre d'activation aura tendance à aussi diminuer ;
- Si les valeurs $P - P^0$ et $P + P^0$ augmentent alors le coût de production variable aura tendance à aussi augmenter ;
- K : c'est le nombre minimum de fois qu'il faut faire de le plein de la citerne pour satisfaire toutes les recharges. si cette valeur augmente alors le coût de production aura tendance à aussi augmenter ;
- Mu : c'est la quantité totale d'hydrogène à recharger. si cette valeur augmente alors le coût de production aura tendance à aussi augmenter.

Description de l'architecture du réseau de neurones INDIC_COUT

Notre but est de produire le plus tôt possible au meilleur coût donc on aimerait que les meilleurs coûts soit plutôt au début de l'espace temps pour pouvoir finir la tournée le plus tôt possible. On veut calculer des indicateurs qu'on manipulera à travers des équations pour faire une approximation du coût de production $COST$. On va écrire ces équations en respectant le sens des indicateurs, ils seront positifs ou négatifs en fonction de l'impact qu'ils ont sur $COST$. Les coefficients des équations seront trouvés à l'aide d'un réseau de neurones.

Dans un premier temps, on veut faire une approximation du coût $Cost_A$ de chaque activation de la micro-usine, le coût total d'activation est environ $Cost_A \times K$, où K est le nombre minimum de fois qu'il faut faire le plein de la citerne pour satisfaire toutes les recharges. Ta doit exprimer la pression qu'on a sur le nombre d'activation minimal. On sait qu'au cours du procédé on aura au moins une activation. d'où $1 + Ta$. La valeur $1 + Ta$ est un coefficient multiplicateur qu'on appliquera au coût d'activation. Étant donné que dans notre problème A vaut le coût d'activation et que $A^0 = 0$, la valeur $Cost_A = (1 + Ta) \times A$.

Dans un second temps, on veut faire une approximation du coût $Cost_P$ par unité de production, le coût de production sera environ $Cost_P \times Mu$, où Mu est la quantité totale d'hydrogène à recharger.

Le calcul de $Cost_P$ dépend de l'intervalle $[0, 2P]$. Si α_1 et α_2 se rapprochent de 1 alors le coût $Cost_P$ a tendance à se rapprocher de $2P$. Si α_1 et α_2 se rapprochent de 0 alors le coût $Cost_P$ a tendance à se rapprocher de 0. On va écrire nos équations en respectant le sens des indicateurs, ces derniers seront positifs ou négatifs en fonction de l'impact qu'ils ont sur $Cost_P$ sachant que notre objectif global est qu'il soit le plus petit possible c'est-à-dire qu'il se rapproche de 0. Les indicateurs utilisés pour calculer Z_1 et Z_2 sont positifs lorsqu'ils poussent Z_1 et Z_2 vers la droite (ils deviennent de plus en plus grand). Les indicateurs utilisés pour calculer Z_1 et Z_2 sont négatifs lorsqu'ils poussent Z_1 et Z_2 vers la gauche (ils deviennent de plus en plus petit). Plus les valeurs positives de Z_1 et Z_2 augmentent plus $Cost_P$ se rapproche de 0. Et, plus les valeurs négatives de Z_1 et Z_2 augmentent plus $Cost_P$ se rapproche de $2P$.

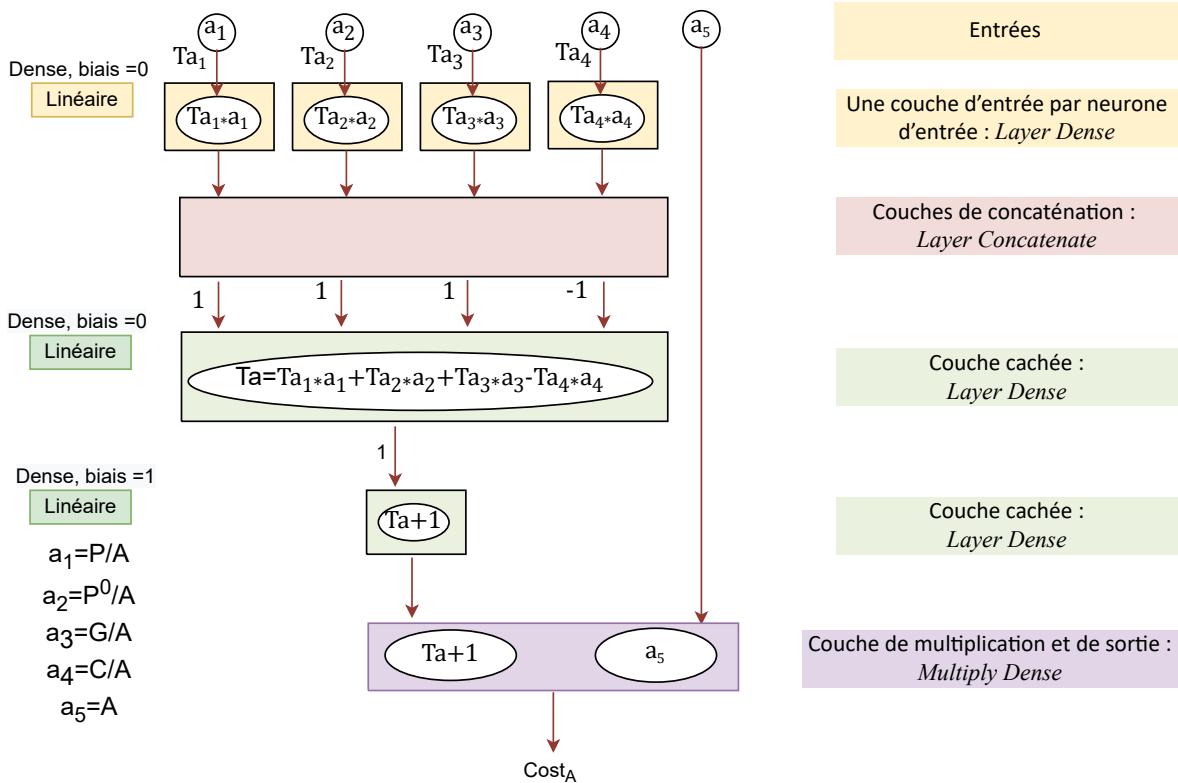
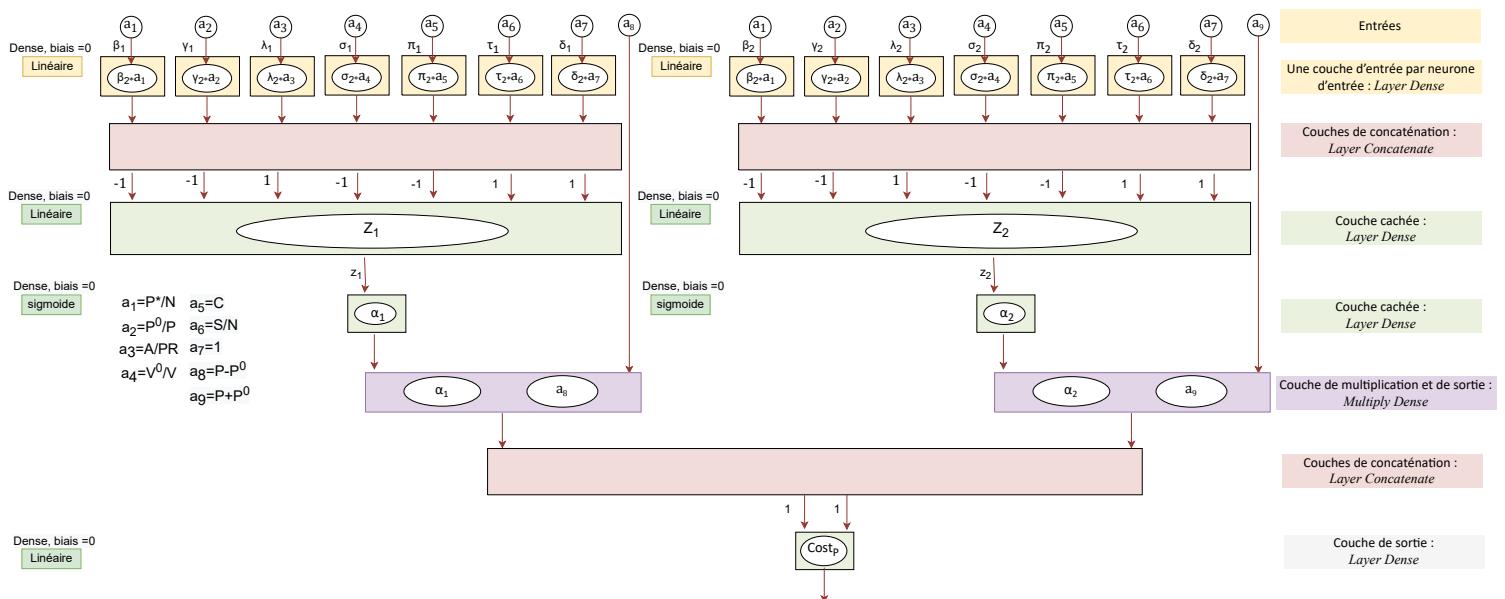
Nous voulons obtenir une approximation du coût de production $COST$. Donc nous définissons :

$$COST = K.Cost_A + Mu.Cost_P ;$$

- $Cost_A = (1 + Ta).A$
 - $Ta = (1/A).(Ta_1.P + Ta_2.P^0 + Ta_3.G - Ta_4.C)$, avec $Ta_1, Ta_2, Ta_3, Ta_4 \geq 0$;
- $Cost_P = \alpha_1.(P - P^0) + \alpha_2.(P + P^0)$ avec $\alpha_1, \alpha_2 \in [0, 1]$;
 - $\alpha_1 = \Phi^{z1}(Z_1)$;
 - $Z_1 = -\beta_1.(P^*/N) - \gamma_1.(P^0/P) + \lambda_1.(A/PR) - \sigma_1.(V^0/V) - \pi_1.C + \tau_1.(S/N) + \delta_1$, où $\beta_1, \gamma_1, \lambda_1, \sigma_1, \pi_1, \tau_1 \geq 0$ et δ_1 est non signé.
 - $\alpha_2 = \Phi^{z2}(Z_2)$;
 - $Z_2 = -\beta_2.(P^*/N) - \gamma_2.(P^0/P) + \lambda_2.(A/PR) - \sigma_2.(V^0/V) - \pi_2.C + \tau_2.(S/N) + \delta_2$, où $\beta_2, \gamma_2, \lambda_2, \sigma_2, \pi_2, \tau_2 \geq 0$ et δ_2 est non signé.

A partir de ces équations, nous construisons les réseaux de neurones des figures (7.5) et (7.6). Ces réseaux de neurones seront fusionnés (Voir figure 7.7) afin d'obtenir le réseau de neurones qui va prédire la valeur $COST$. Les sorties des réseaux de neurones des figures (7.5) et (7.6) seront les entrées d'une couche de multiplication avec les valeurs K et Mu . Les sorties de cette précédente couche seront les entrées d'un layer Dense et la sortie de ce layer Dense sera la valeur $COST$. Les poids de ce layer Dense seront tous fixés à 1. L'architecture du réseau de neurones de cette partie est la suivante :

- Elle a huit couches d'entrées constituées chacune d'un neurone ;
- Elle a sept couches cachées ;
- Elle a une couche de sortie d'un neurone.

FIGURE 7.5 – Réseau de neurones prédisant le coût $Cost_A$.FIGURE 7.6 – Réseau de neurones prédisant le coût $Cost_P$.

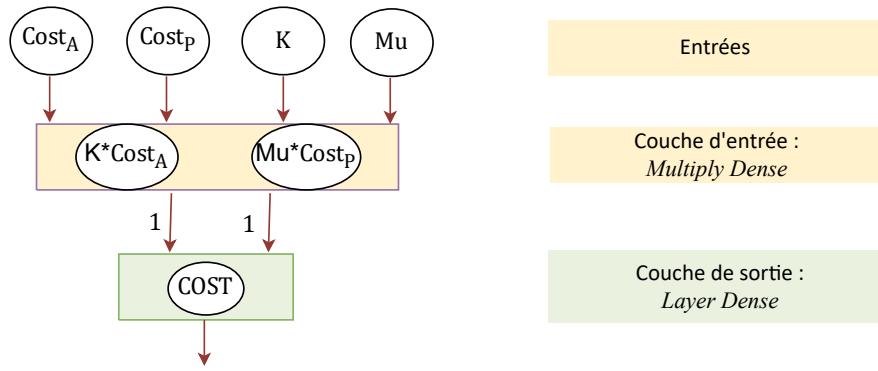


FIGURE 7.7 – Réseau INDIC_COUT prédisant le coût de production $COST$.

Les fonctions d’activation utilisées sont la fonction sigmoïde et la fonction linéaire. Le nombre de poids et de biais du réseau de neurones INDIC_COUT ne dépend pas de la taille des instances. L’algorithme d’optimisation utilisé est la descente de gradient stochastique. Les données de validation sont 1/3 des données d’apprentissage. Le nombre d’itérations correspondant à l’itération qui a fourni le plus petit gap sur les données de test est 201. La taille des batchs fixée ici est 32.

Dans le réseau INDIC_COUT, on a 20 poids à apprendre.

Description des sorties du réseau de neurones INDIC_COUT

La sortie du réseau de neurones INDIC_COUT est la valeur du coût de production $COST$.

7.4 Expérimentations numériques

Dans cette section dédiée aux expérimentations numériques, on va d’abord présenter les objectifs et le contexte technique de nos expériences. Ensuite, on présente les instances utilisées pour entraîner et tester nos réseaux. On finit cette section en présentant les résultats des tests effectués sur les réseaux présentés ci-dessus.

7.4.1 Objectifs et contexte technique

Ce que nous voulons ici, c’est obtenir une évaluation des performances des réseaux de neurones des figures (7.1), (7.2), (7.3), (7.4) et (7.7). Cela consiste à comparer les valeurs prédites par les réseaux aux valeurs optimales.

Les réseaux de neurones ont été implémentés en Python, sur un ordinateur fonctionnant sous le système d’exploitation Windows 10 avec un processeur IntelCore i5-6500@3.20 GHz, 16 Go de RAM et l’IDE utilisé est Visual Studio 2019. On a utilisé Tensorflow/Keras pour implémenter et manipuler nos réseaux de neurones.

Les graphiques des gaps montrent la variation de l’erreur des données d’apprentissage et des données de test lorsqu’on fait varier le nombre d’itérations lors de la phase d’entraînement du réseau. La courbe rouge représente l’évolution de l’erreur calculée pour le groupe de test et la courbe bleue représente l’erreur calculée pour le groupe apprentissage. Nous ne montrerons pas ici la courbe des gaps des données de validation car elle n’est pas pertinente pour juger de la qualité de nos réseaux.

7.4.2 Instances

On a utilisé la procédure de génération d'instance décrite à la section (4.7.2) pour générer 6000 instances du problème SMEPC dont le nombre de périodes est égal à 20. Pour obtenir nos instances, on a élaboré un générateur d'instances. Les paramètres qu'on a utilisé pour générer aléatoirement nos instances sont :

1. Concernant le nombre de stations, les instances peuvent être subdivisées en 4 catégories : les instances de 8 stations, celles de 10 stations, celles de 15 stations et celles de 20 stations. Le nombre de stations a été choisi arbitrairement.
2. Concernant les coordonnées des stations, les instances peuvent être subdivisées en 3 catégories en fonction du carré dans lequel les coordonnées des instances sont sélectionnées aléatoirement : on a les instances dont les coordonnées de stations sont sélectionnées dans un carré 10×10 , celles sélectionnées dans un carré 20×20 et celles sélectionnées dans un carré 30×30 .
3. Concernant les coûts fixes et les coûts variables, on a décidé de générer ces coûts aléatoirement dans certains intervalles. Les instances ici peuvent être subdivisées en 5 catégories :
 - On a les instances dont le coût fixe est 5 et les coûts variables appartiennent à l'intervalle $[1, 10]$. Pour cette catégorie d'instances même si le nombre de stations augmente, le coût de production aura du mal à augmenter car les coûts de production sont très petits ;
 - On a les instances dont le coût fixe est 50 et les coûts variables appartiennent à l'intervalle $[20, 40]$;
 - On a les instances dont le coût fixe est 100 et les coûts variables appartiennent à l'intervalle $[20, 40]$;
 - On a les instances dont le coût fixe est 100 et les coûts variables appartiennent à l'intervalle $[50, 100]$;
 - On a les instances dont le coût fixe est 150 et les coûts variables appartiennent à l'intervalle $[50, 100]$.

En croisant les critères 1, 2 et 3 ci-dessus, on obtient $4 \times 3 \times 5 = 60$ possibilités et pour chacune on génère 100 instances. Ce qui nous permet d'obtenir nos 6000 instances.

A partir de chaque instance, on calcule les valeurs des indicateurs qui serviront à avoir les valeurs utilisées comme entrées des réseaux de neurones. On récupère après exécution de l'algorithme Pipeline VD_PM du chapitre précédent pour chaque instance le numéro de période de la dernière opération de recharge et le coût de production. Ces données sont utilisées pour entraîner et tester nos réseaux de neurones. Les 6000 instances seront séparées en deux catégories d'instance à savoir les instances d'apprentissage qui nous serviront à entraîner les réseaux et les instances de test qui nous serviront à analyser le comportement de nos réseaux sur de nouvelles données. On décide d'avoir 90 instances de test et 5910 instances d'apprentissage. Pour avoir une idée des caractéristiques des instances utilisées, on fait une analyse statistique de nos instances. La section suivante est consacrée à présenter cette analyse.

Analyse statistique des instances

Les figures (7.10), (7.11), (7.12), (7.9) et (7.13) illustrent l'analyse statistique des instances. Par souci de lisibilité de nos figures, on n'affiche pas les instances une par une mais on affiche la moyenne

de chaque paquet de 100 instances. On ordonne préalablement nos instances par coût de production croissant. Les instances générées avec le coût fixe 5 et les coûts variables appartenant à l'intervalle [1, 10] ont des coûts tellement petits que le nombre de stations n'impacte pas trop le coût d'où les pics observés sur les figures (7.10) (b), (7.11) (b), (7.12) (b)

La figure (7.8) représente le coût moyen de production regroupé par paquets de 100 instances. Sachant que toutes les instances ont préalablement été ordonnées par coût de production croissant. On constate que les coûts de production de nos instances sont compris entre 0 et 2000.

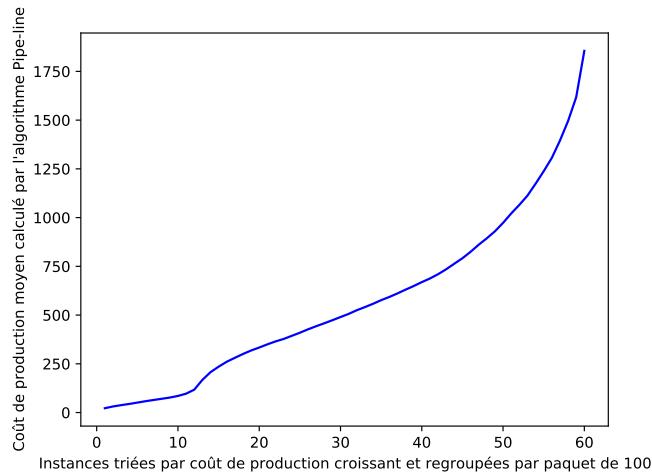


FIGURE 7.8 – Coût moyen de production par ordre croissant et regroupé par paquet de 100 instances. Ce coût a été obtenu en exécutant l'algorithme Pipe-line VD_PM du chapitre précédent.

La figure 7.9(a) représente le coût fixe moyen de production regroupé par paquet de 100 instances. La figure 7.9(b) représente le coût variable moyen de production regroupé par paquet de 100 instances. Sachant que toutes les instances ont préalablement été ordonnées par coût de production croissant. De ces figures, on peut dire que le coût de production croît lorsque le cout fixe et le coût variable croient aussi. Ce qui montre que ces coûts peuvent être utilisés comme entrées des réseaux qui prédisent le coût de production.

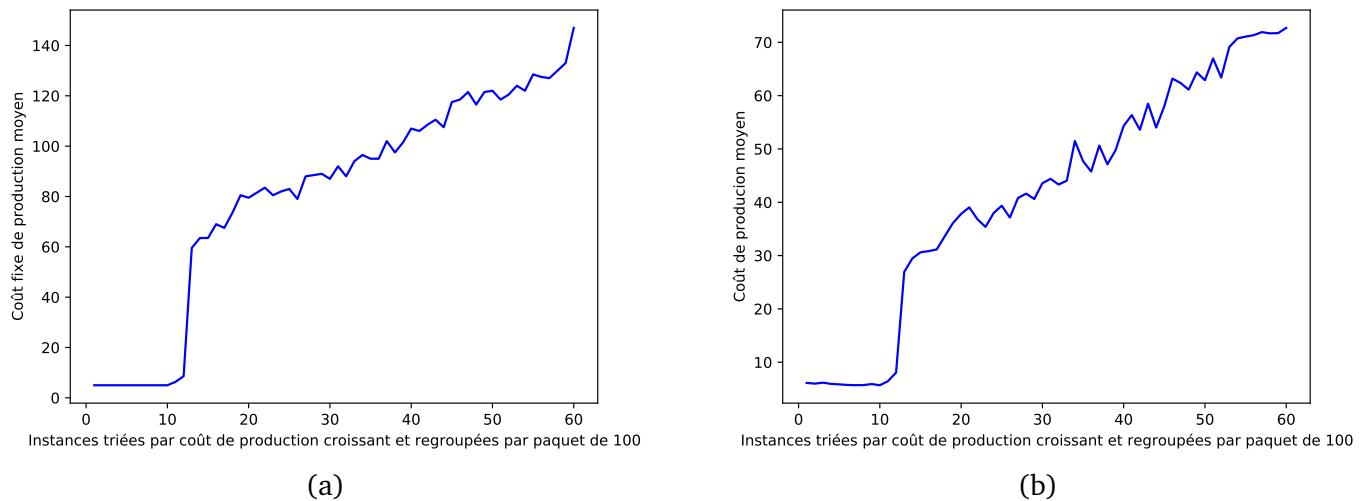


FIGURE 7.9 – La figure (a) représente le coût fixe moyen de production regroupé par paquet de 100 instances. La figure (b) représente le coût variable moyen de production regroupé par paquet de 100 instances. Sachant que les instances ont préalablement été classées par coût de production croissant.

La figure (7.10)(a) représente le nombre d'instances par nombre de stations. On constate qu'on a 1500 instances de 8 stations, 1500 instances de 10 stations, 1500 instances de 15 stations et 1500 instances de 20 stations. La figure (7.10)(b) représente la moyenne du nombre de stations regroupé par paquet de 100 instances. On constate que du paquet d'instances 1 au paquet d'instances 12, le nombre de stations croît au fur et à mesure que le coût de production croît. Mais à partir du paquet d'instances 13 on a une brusque chute du nombre de stations moyen malgré le fait que le coût augmente. Ceci est dû au fait qu'à la génération des instances, on a créé des instances avec un coût fixe très petit 5, alors que les autres coûts sont au moins quatre fois plus élevés. On peut conclure que le nombre de stations n'est pas une valeur pertinente à utiliser pour prédire le coût de production.

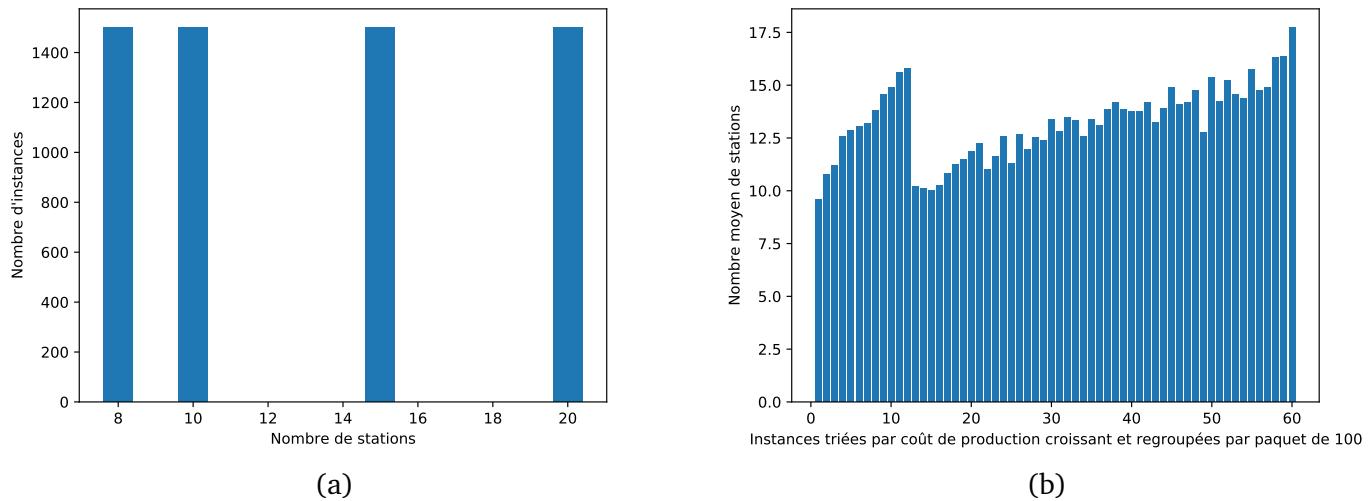


FIGURE 7.10 – La figure (a) représente le nombre d'instances par nombre de stations. La figure (b) représente la moyenne du nombre de stations regroupé par paquet de 100 instances. Sachant que les instances ont été classées par coût de production croissant.

On a exécuté l'algorithme DPS_VD du chapitre précédent pour obtenir la stratégie de recharge

réduite. La figure (7.11)(a) représente le nombre d'instances par nombre de recharges. On constate que la majorité des instances ont entre 4 et 6 recharge. La figure (7.11)(b) représente la moyenne du nombre de recharges regroupé par paquet de 100 instances. Sachant que toutes les instances ont préalablement été ordonnées par coût de production croissant. On constate que du paquet d'instances 1 au paquet d'instances 12, le nombre moyen de recharges croît au fur et à mesure que le coût de production croît. Mais à partir du paquet d'instances 13 on a une brusque chute du nombre moyen de recharges malgré le fait que le coût augmente. Ceci est dû au fait qu'à la génération des instances, on a créé des instances avec un coût fixe très petit 5, alors que les autres coûts sont au moins quatre fois plus élevés. On peut conclure que le nombre moyen de recharges n'est pas une valeur pertinente à utiliser pour prédire le coût de production.

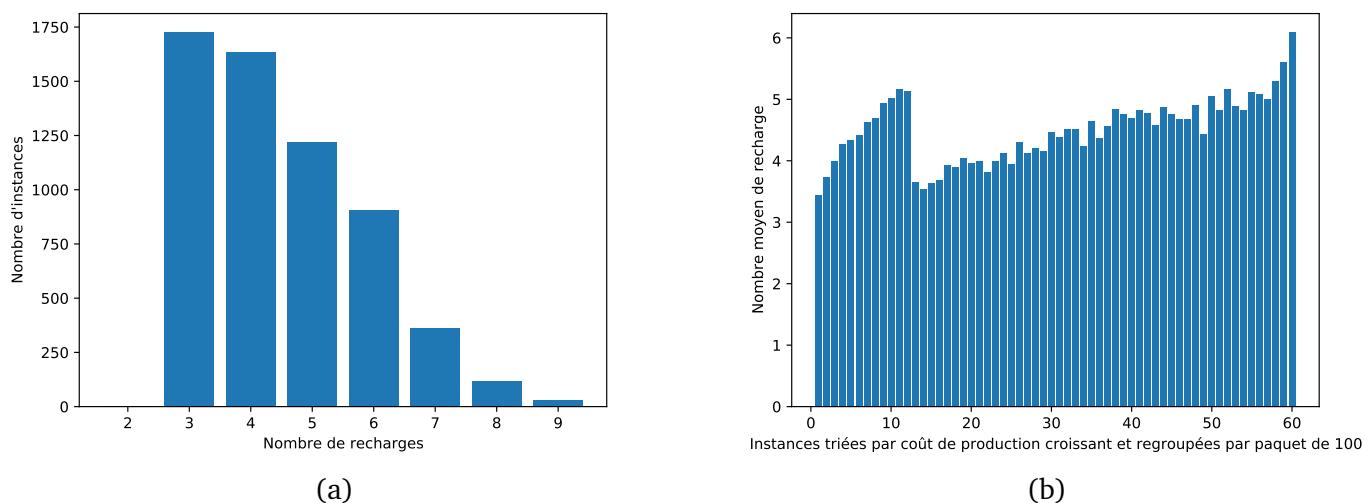


FIGURE 7.11 – La figure (a) représente le nombre d'instances par nombre de recharges. La figure (b) représente la moyenne du nombre de recharges regroupé par paquet de 100 instances. Sachant que les instances ont été classées par coût de production croissant.

La figure (7.12) (a) représente la durée moyenne d'une période de production regroupé par paquet de 100 instances. Sachant que toutes les instances ont préalablement été ordonnées par coût de production croissant. La figure (7.12) (b) représente l'évolution du temps maximal moyen ($TMax$) regroupé par paquet de 100 instances. Sachant que toutes les instances ont préalablement été ordonnées par coût de production croissant. On peut conclure que la durée moyenne d'une période et le temps maximal moyen ne sont pas des valeurs pertinentes à utiliser pour prédire le coût de production.

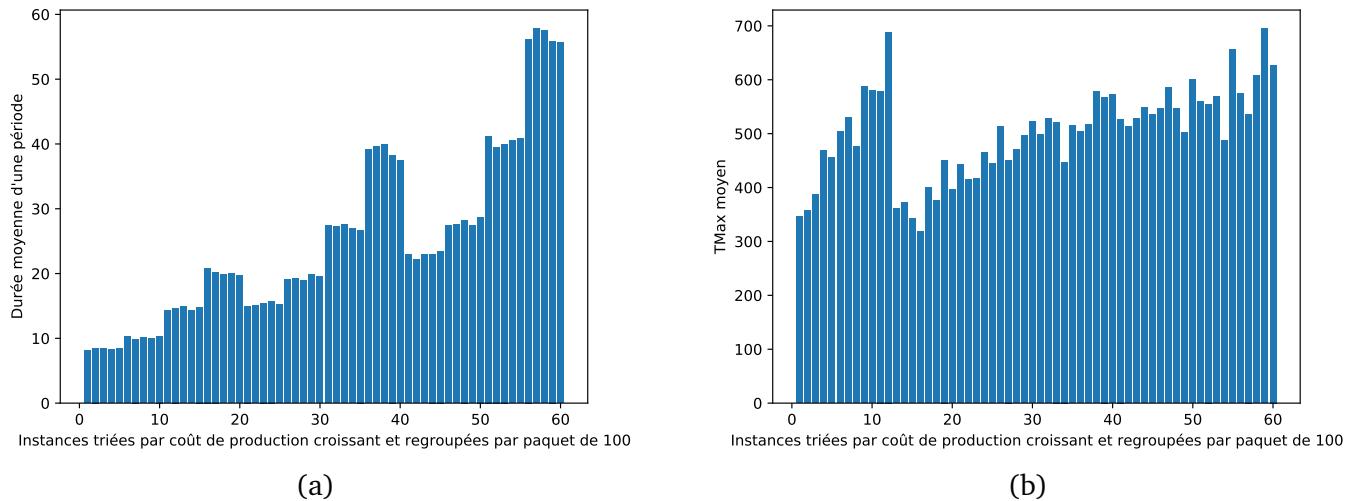


FIGURE 7.12 – La figure (a) représente la durée moyenne d'une période de production regroupé par paquet de 100 instances. La figure (b) représente l'évolution du temps maximal moyen regroupé par paquet de 100 instances. Sachant que les instances ont été classées par coût de production croissant.

La figure (7.13)(a) représente la capacité moyenne de la citerne regroupé par paquet de 100 instances. La figure (7.13)(b) représente la capacité moyenne du réservoir regroupé par paquet de 100 instances. Sachant que les instances ont été classées par coût de production croissant. On peut conclure que la capacité moyenne de la citerne et la capacité moyenne du réservoir ne sont pas des valeurs pertinentes à utiliser pour prédire le coût de production.

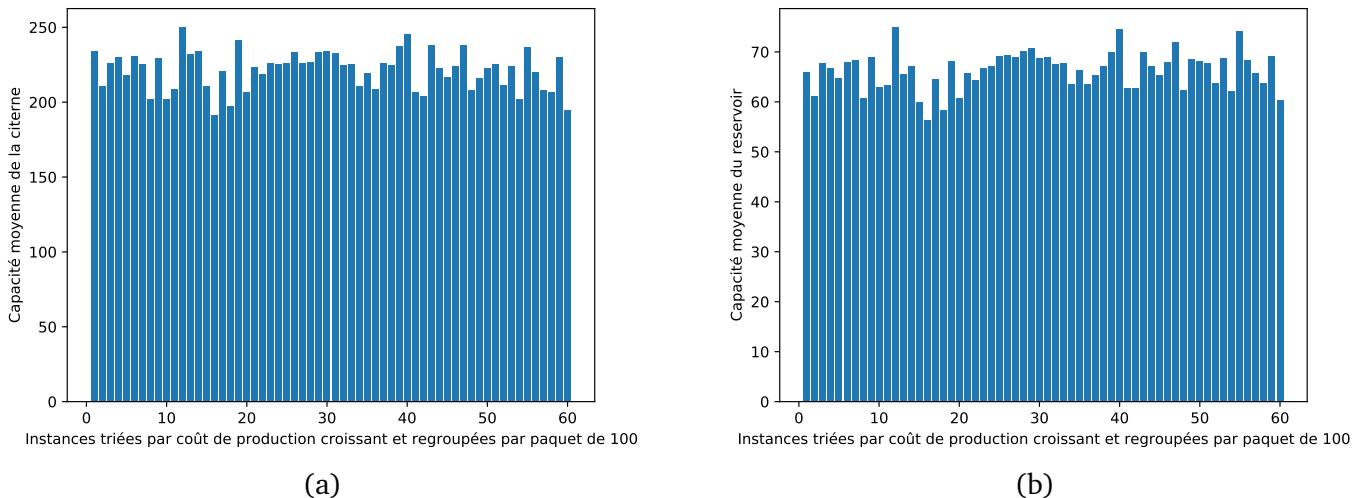


FIGURE 7.13 – La figure (a) représente la capacité moyenne de la citerne regroupé par paquet de 100 instances. La figure (b) représente la capacité moyenne du réservoir regroupé par paquet de 100 instances. Sachant que les instances ont été classées par coût de production croissant.

7.4.3 Moyenne de la valeur absolue des gaps

Le but de cette section est de présenter les gaps obtenus après entraînement de nos réseaux de neurones. La métrique d'évaluation de nos réseaux est le gap entre la valeur V_{Opt} utilisée pour entraîner nos réseaux et la valeur V_{Predit} prédict par nos réseaux : $(|V_{Opt} - V_{Predit}|)/V_{Opt}$. On présente ici la moyenne des gaps des données d'apprentissage et des données de test. On trouve au tableau (7.3) les

résultats de nos réseaux qui prédisent le coût de production. On trouve au tableau (7.4) les résultats de nos réseaux qui prédisent le numéro de période de la dernière recharge. La signification de chaque colonne est :

- La colonne nommée « Nom du réseau de neurones » est le nom du réseau de neurones ;
- La colonne nommée « fig. réseau » donne les numéros des figures qui illustrent l'architecture du réseau de neurones ;
- La colonne nommée « fig. résultats » donne le numéro de figures qui illustrent la courbe des valeurs prédites par le réseau de neurones, la courbe des valeurs utilisées pour entraîner le réseau de neurones et les courbes de l'évolution du gap au fil des itérations ;
- La colonne nommée « # itérations » est le nombre d'itérations ;
- La colonne nommée « Gap test » est la moyenne en pourcentage des gaps des données de test ;
- La colonne nommée « Gap entraînement » est la moyenne en pourcentage des gaps des données d'entraînement ;
- La colonne nommée « # poids » est le nombre de poids synaptiques des réseaux de neurones.

Nom du réseau de neurones	fig. réseau	fig. résultats	# itérations	Gap test(%)	Gap entraînement(%)	# poids
Réseau SIMPLE_TYPE	(7.1)	(7.14) (7.15)	20	25,45	24,8	697
Réseau SIMPLE_PERIODE	(7.2)	(7.16) (7.17)	2	23,28	38,03	729
Réseau MIXTE_COUT	(7.3)	(7.18) (7.19)	161	17,14	14,49	570
Réseau INDIC_COUT	(7.5)(7.6) (7.7)	(7.20) (7.21)	201	24,19	13,27	20

TABLE 7.3 – Apprentissage du coût de production : Moyenne de la valeur absolue des gaps des données de test et d'apprentissage.

Nom du réseau de neurones	fig. réseau	fig. résultats	# itérations	Gap test (%)	Gap entraînement(%)	# poids
Réseau MIXTE_TEMPS	(7.3)	(7.22) (7.23)	195	11,26	8,31	570
Réseau INDIC_TEMPS	(7.4)	(7.24) (7.25)	3	13,21	8,78	9

TABLE 7.4 – Apprentissage du numéro de période de la dernière recharge : Moyenne de la valeur absolue des gaps des données de test et d'apprentissage.

Notre objectif en terme de gap lorsqu'on construisait nos réseaux de neurones était d'être en-dessous de 5%, ce qui est la plupart du temps le cas d'une bonne heuristique en recherche opérationnelle. Autrement dit, on espérait ne pas être très loin des résultats d'une heuristique mais ce n'est malheureusement pas le cas. Les gaps des réseaux SIMPLE_TYPE et SIMPLE_PERIODE sont les plus élevés, peut-être est-ce à cause de la « simplicité » de ces réseaux. On est assez déçu des résultats des réseaux MIXTE_COUT et MIXTE_TEMPS. Les gaps du réseau INDIC_COUT ne sont pas très décevants au vu du faible nombre de poids de ce réseau. Cela est peut-être dû au fait qu'on a peu d'indicateurs.

La capacité de généralisation d'un réseau est sa capacité à faire de bonnes prédictions sur de nouvelles données. La capacité de généralisation se mesure ici à l'aide des gaps des données de

test. On constate ici que les réseaux qui ont la meilleure capacité de généralisation sont les réseaux **MIXTE_COUT** et **MIXTE_TEMPS**. Tandis que les réseaux qui ont un gap d'entraînement le plus faible sont les réseaux **INDIC_COUT** et **MIXTE_TEMPS**.

7.4.4 Résultats des réseaux SIMPLE_TYPE et SIMPLE_PERIODE

La fonction d'activation permet de changer la façon dont une valeur est représentée. Nous allons tester plusieurs fonctions d'activation sur le réseau **SIMPLE_PERIODE**. La liste de fonctions qu'on testera est la suivante :

- La fonction linéaire : $f(x) = x$, elle laisse passer toutes les valeurs dans les couches suivantes.
- La fonction relu (Rectified Linear Unit) :

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0. \end{cases}$$

Elle ne laisse passer dans les couches suivantes que les valeurs positives.

- La fonction elu (Exponential Linear Unit) :

$$f(x) = \begin{cases} \alpha \times (e^x - 1) & \text{si } x < 0 \\ x & \text{si } x \geq 0. \end{cases}$$

Elle est une amélioration de la fonction RELU car elle lisse aussi les valeurs négatives

- La fonction selu (Scaled Exponential Linear Unit) est une amélioration de la fonction elu.

$$f(x) = \begin{cases} scale \times \alpha \times (e^x - 1) & \text{si } x < 0 \\ scale \times x & \text{si } x \geq 0. \end{cases}$$

Les valeurs utilisées sont les valeurs par défaut $\alpha = 1,67326324$, $scale = 1,05070098$.

- La fonction softplus : $f(x) = \log(e^x + 1)$, est une approximation de la fonction relu.
- La fonction sigmoïde : $f(x) = \frac{1}{1 + e^{-x}}$ elle donne une valeur dans l'intervalle [0,1].
- La fonction tangente hyperbolique $f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$ elle donne une valeur dans l'intervalle [-1,1].

Le tableau (7.5) présentent les gaps obtenus avec le réseau **SIMPLE_PERIODE** lorsqu'on modifie les fonctions d'activation de la couche d'entrées et de la couche cachée. La signification des colonnes est :

- La colonne « Activation couche 1 » donne le nom de la fonction d'activation utilisée pour la couche d'entrées du réseau **SIMPLE_PERIODE**
- La colonne « Activation couche 2 » donne le nom de la fonction d'activation utilisée pour la couche cachée du réseau **SIMPLE_PERIODE**
- La colonne nommée « Gap test » est la moyenne en pourcentage des gaps des données de test ;
- La colonne nommée « Gap entraînement » est la moyenne en pourcentage des gaps des données d'entraînement ;

On constate que le fait de changer la façon dont les données d'entrées sont classées a une influence sur les résultats de nos réseaux. De plus, on constate aussi que le fait de changer de fonction d'activation a une influence sur les résultats de nos réseaux. On constate que c'est la fonction selu qui nous donne le meilleur gap sur les données de test. C'est donc cette fonction que nous utiliserons pour tester les réseaux **SIMPLE_TYPE** et **SIMPLE_PERIODE**.

Activation couche 1	Activation couche 2	Gap entraînement (%)	Gap test (%)
linéaire	linéaire	89,08	188,34
tangente h.	tangente h.	67,10	41,88
sigmoïde	sigmoïde	66,68	40,99
relu	relu	21,52	33,47
relu	selu	21,54	28,75
selu	relu	17,69	27,86
softplus	softplus	20,03	21,63
elu	elu	19,95	21,67
selu	selu	20	21,14

TABLE 7.5 – Comparaison des fonctions d'activation du réseau **SIMPLE_PERIODE**.

Résultats du réseau **SIMPLE_TYPE**

La figure (7.14) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,200] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. On constate que le gap ici se stabilise vite. Au bout d'environ 25 itérations, on a déjà un gap assez stable, mais après 100 itérations les gaps redeviennent un peu instables. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

Les résultats du réseau **SIMPLE_TYPE** sont illustrés par les figures (7.15). On veut voir la corrélation entre les valeurs prédites par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L'axe des abscisses est respectivement pour la figure (7.15) (a) et (7.15) (b) les instances d'apprentissage triées par coût de production croissant et regroupées par paquet de 100 et les instances de test triées par coût de production croissant. L'axe des ordonnées est respectivement pour la figure (7.15) (a) et (7.15) (b) le coût de production moyen et le coût de production. La courbe verte est la courbe des valeurs optimales et la courbe rouge est la courbe des valeurs prédites par le réseau de la figure (7.1). (a) représente les données d'apprentissage et (b) représente les données de test. En observant la courbe (a), on a l'impression que le réseau fait de bonnes prédictions mais c'est trompeur car on a fait des paquets de 100 instances et on a fait des moyennes sur ces paquets, ce qui a pour conséquence de compenser les erreurs. Le fait de faire des moyennes a lissé la courbe des valeurs prédites de la figure (7.15) (a) sinon cette dernière se présente aussi en dents de scie. On observe que la courbe des valeurs prédites a l'allure de la courbe des valeurs optimales avec une tendance à être légère en dessous de celle-ci après environ l'itération 45 et à être au dessus avant cette itération.

On constate que le réseau **SIMPLE_TYPE** est de mauvaise qualité, ceci peut s'expliquer par le fait que ce réseau est « simple », c'est-à-dire qu'il n'épouse pas la logique de notre problème.

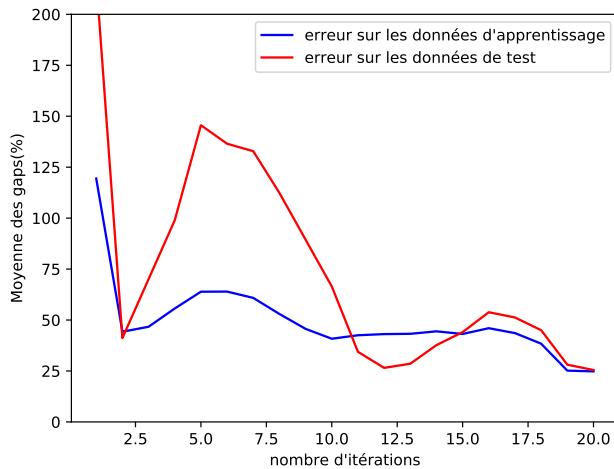


FIGURE 7.14 – Cette figure représente l'évolution du gap lorsqu'on fait varier le nombre d'itérations lors de l'apprentissage du réseau **SIMPLE_TYPE**. Ici, les données d'entrées sont classées par type. La courbe bleue est l'évolution de l'erreur sur les données d'apprentissage et la courbe rouge est l'évolution de l'erreur sur les données de test. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

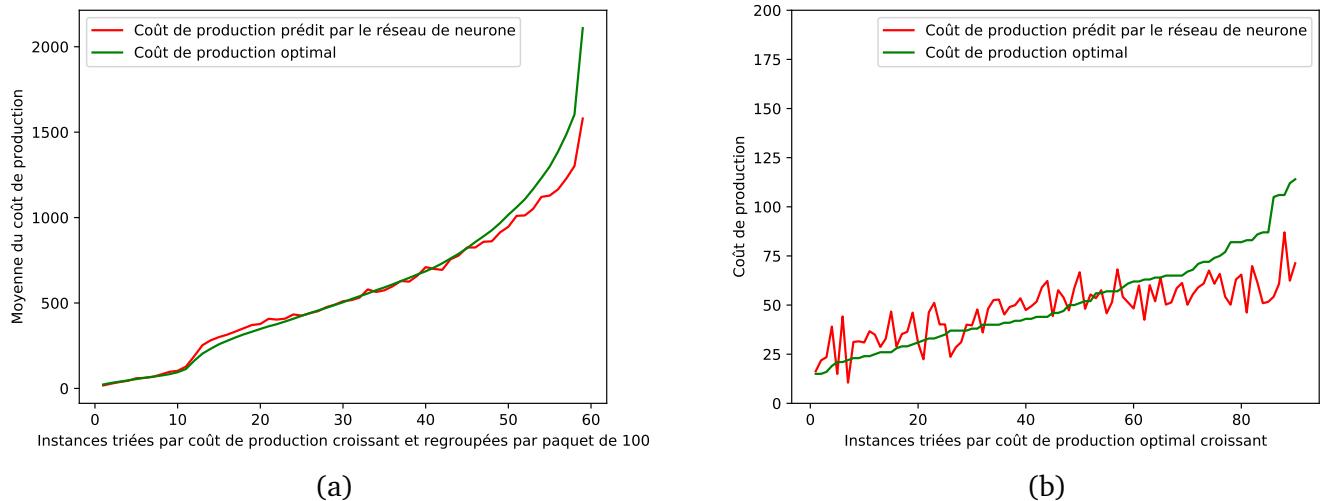


FIGURE 7.15 – Résultats du réseau de neurones **SIMPLE_TYPE** pour la prédiction du coût de production avec des données classées type par type. (a) représente les données d'apprentissage et (b) représente les données de test. La courbe rouge est la courbe des valeurs prédites et la courbe verte est la courbe des valeurs optimales.

Résultats du réseau SIMPLE_PERIODE

La figure (7.16) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,200] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. On constate que le gap sur les données d'apprentissage se stabilise vite, au bout d'environ 10 itérations, on a déjà un gap assez stable. Tandis que le gap sur

les données de test se stabilise au bout d'environ 185 itérations. Le nombre d'itérations maximal correspond à l'itération qui a fournit le plus petit gap sur les données de test.

Les résultats du réseau **SIMPLE_PERIODE** sont illustrés par les figures (7.17). On veut voir la corrélation entre les valeurs prédites par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L'axe des abscisses représente respectivement pour la figure (7.17) (a) et (7.17) (b) les instances d'apprentissage triées par coût de production croissant et regroupées par paquet de 100 et les instances de test triées par coût de production croissant. L'axe des ordonnées est respectivement pour la figure (7.17) (a) et (7.17) (b) le coût de production moyen et le coût de production. La courbe verte est la courbe des valeurs optimales et la courbe rouge est la courbe des valeurs prédites par le réseau de la figure (7.2). La figure (a) représente les données d'apprentissage et la figure (b) représente les données de test. En observant les courbes (a), on a l'impression que le réseau fait de bonnes prédictions mais c'est trompeur car on a fait des paquets de 100 instances et on a fait des moyennes sur ces paquets, ce qui a pour conséquence de compenser les erreurs. Le fait de faire des moyennes a lissé la courbe des valeurs prédites de la figure (7.17) (a) sinon cette dernière se présente aussi en dents de scie. On observe que la courbe des valeurs prédites a l'allure de la courbe des valeurs optimales avec une tendance à être légère en dessous de celle-ci après environ l'itération 43 et à être au dessus avant cette itération.

On constate que le réseau **SIMPLE_PERIODE** est de mauvaise qualité, ceci s'explique par le fait que ce réseau est « simple », c'est-à-dire qu'il n'épouse pas la logique de notre problème.

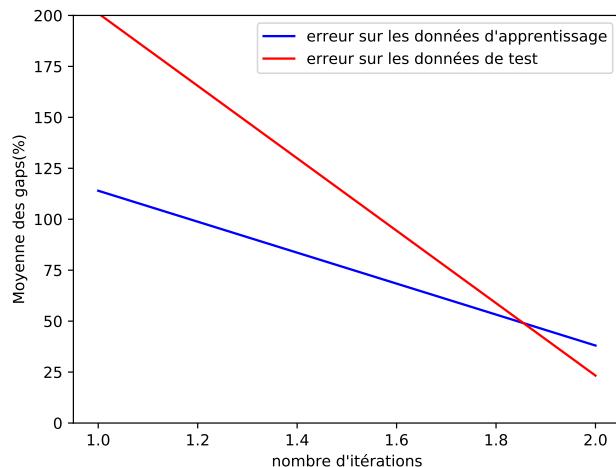


FIGURE 7.16 – Cette figure représente l'évolution du gap lorsqu'on fait varier le nombre d'itérations lors de l'apprentissage du réseau **SIMPLE_PERIODE**. Ici, les données d'entrées sont classées par période. La courbe bleue est l'évolution de l'erreur sur les données d'apprentissage et la courbe rouge est l'évolution de l'erreur sur les données de test. Le nombre d'itérations maximal correspond à l'itération qui a fournit le plus petit gap sur les données de test.

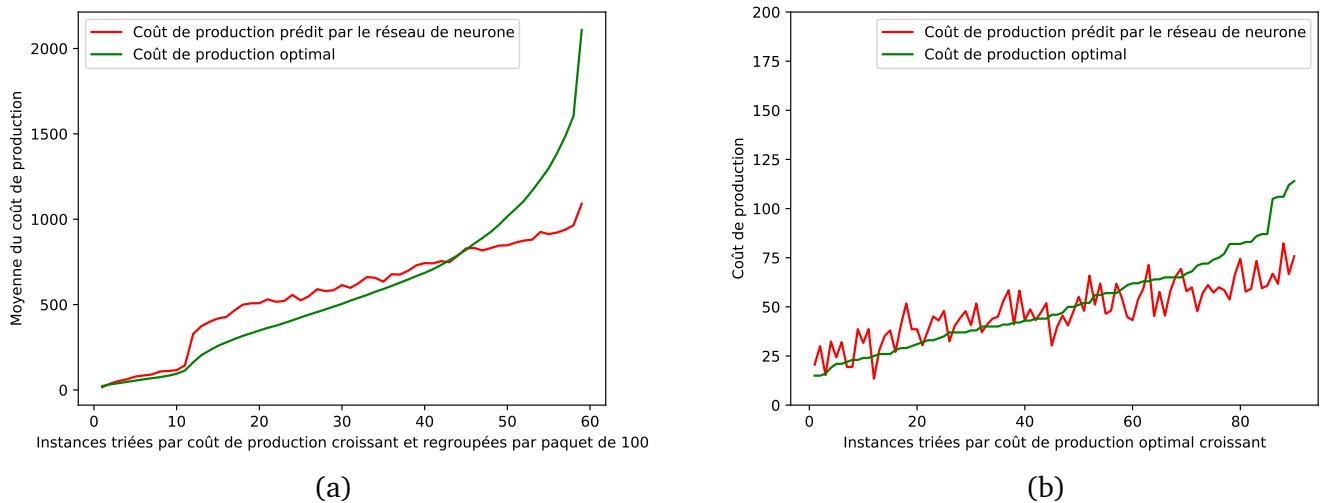


FIGURE 7.17 – Résultats du réseau **SIMPLE_PERIODE** pour la prédiction du coût de production avec des données classées par période. (a) représente les données d'apprentissage et (b) représente les données de test . La courbe rouge est la courbe des valeurs prédites et la courbe verte est la courbe des valeurs optimales.

7.4.5 Résultats du réseau MIXTE_COUT pour la prédiction du coût de production

La figure (7.18) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,50] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. On constate que le gap ici se stabilise vite. Au bout d'environ 50 itérations, on a déjà un gap assez stable. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

Les résultats du réseau **MIXTE_COUT** pour la prédiction du coût de production sont illustrés par les figures (7.19). On veut voir la corrélation entre les valeurs prédites par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L'axe des abscisses est respectivement pour la figure (7.19) (a) et (7.19) (b) les instances d'apprentissage triées par coût de production croissant et regroupées par paquet de 100 et les instances de test triées par coût de production croissant. L'axe des ordonnées est respectivement pour la figure (7.19) (a) et (7.19) (b) le coût de production moyen et le coût de production. La courbe verte est la courbe des valeurs optimales et la courbe rouge est la courbe des valeurs prédites par le réseau de la figure (7.3). La figure (a) représente les données d'apprentissage et la figure (b) représente les données de test. En observant les courbes (a), on a l'impression que le réseau fait de bonnes prédictions mais c'est trompeur car on a fait des paquets de 100 instances et on a fait des moyennes sur ces paquets, ce qui a pour conséquence de compenser les erreurs. Le fait de faire des moyennes a lissé la courbe des valeurs prédites de la figure (7.19) (a) sinon cette dernière se présente aussi en dents de scie. On observe que la courbe des valeurs prédites a l'allure de la courbe des valeurs optimales avec une tendance à être au dessus de cette dernière.

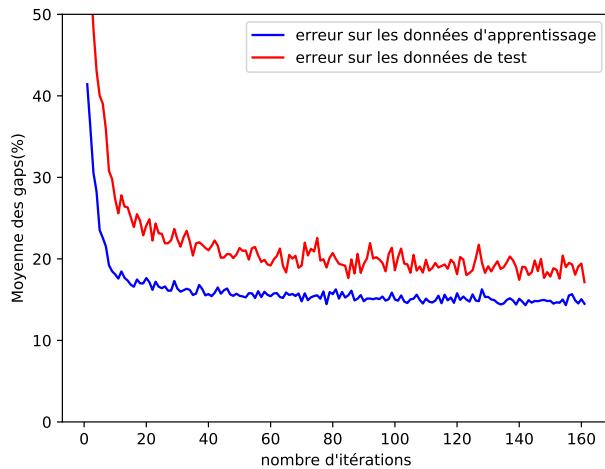


FIGURE 7.18 – Cette figure représente l'évolution du gap lorsqu'on fait varier le nombre d'itérations lors de l'apprentissage du réseau **MIXTE_COUT** pour la prédiction du coût de production. La courbe bleue est l'évolution de l'erreur sur les données d'apprentissage et la courbe rouge est l'évolution de l'erreur sur les données de test. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

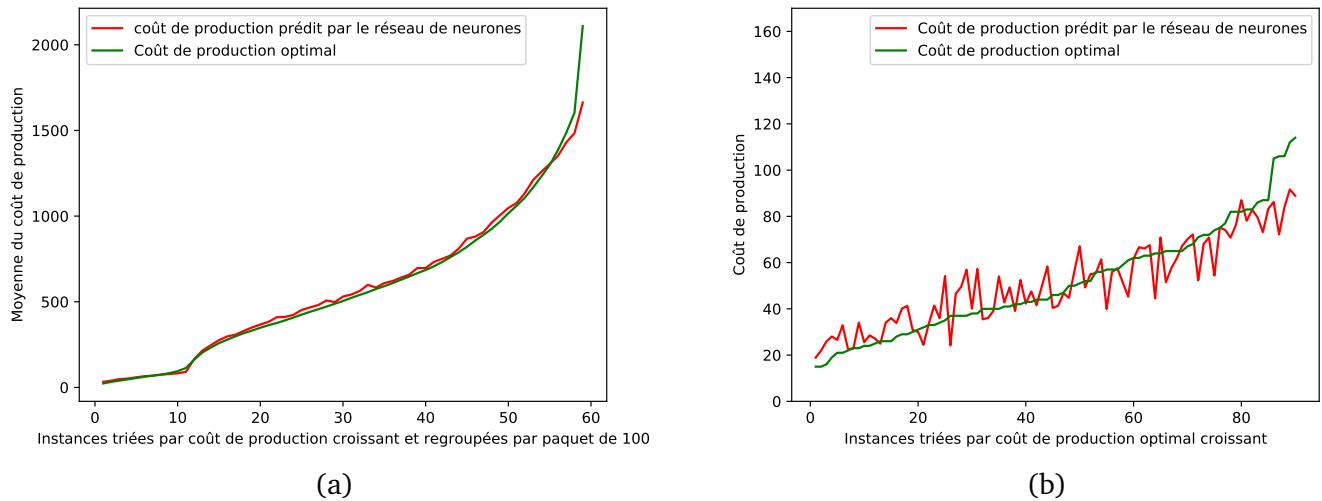


FIGURE 7.19 – Résultats du réseau **MIXTE_COUT** pour la prédiction du coût de production. (a) représente les données d'apprentissage et (b) représente les données de test. La courbe rouge est la courbe des valeurs prédites et la courbe verte est la courbe des valeurs optimales.

7.4.6 Résultats du réseau **INDIC_COUT** pour la prédiction du coût de production

La figure (7.20) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,50] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

Les résultats du réseau **INDIC_COUT** pour la prédiction du coût de production sont illustrés par

les figures (7.21). On veut voir la corrélation entre les valeurs prédictes par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L'axe des abscisses est respectivement pour la figure (7.21) (a) et (7.21) (b) les instances d'apprentissage triées par coût de production croissant et regroupées par paquet de 100 et les instances de test triées par coût de production croissant. L'axe des ordonnées est respectivement pour la figure (7.21) (a) et (7.21) (b) le coût de production moyen et le coût de production. La courbe verte est la courbe des valeurs optimales et la courbe rouge est la courbe des valeurs prédictes par le réseau de la figure (7.7). La figure (a) représente les données d'apprentissage et la figure (b) représente les données de test. En observant les courbes (a), on a l'impression que le réseau fait de bonnes prédictions mais c'est trompeur car on a fait des paquets de 100 instances et on a fait des moyennes sur ces paquets, ce qui a pour conséquence de compenser les erreurs. Le fait de faire des moyennes a lissé la courbe des valeurs prédictes de la figure (7.21) (a) sinon cette dernière se présente aussi en dents de scie. On observe que la courbe des valeurs prédictes a l'allure de la courbe des valeurs optimales avec une tendance à être en dessous de cette dernière.

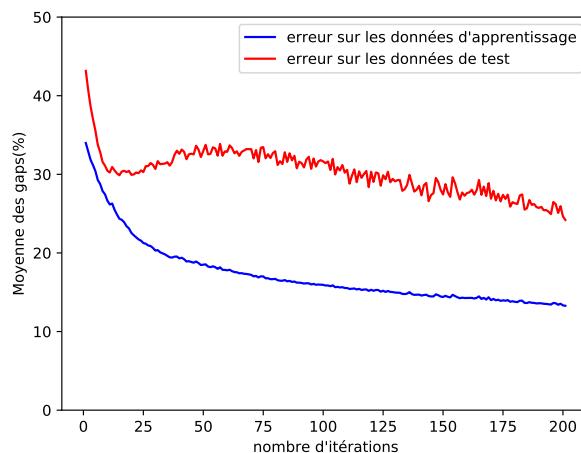


FIGURE 7.20 – Cette figure représente l'évolution du gap lorsqu'on fait varier le nombre d'itérations lors de l'apprentissage du réseau de neurones **INDIC_COUT** pour la prédiction du coût de production. La courbe bleue est l'évolution de l'erreur sur les données d'apprentissage et la courbe rouge est l'évolution de l'erreur sur les données de test. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

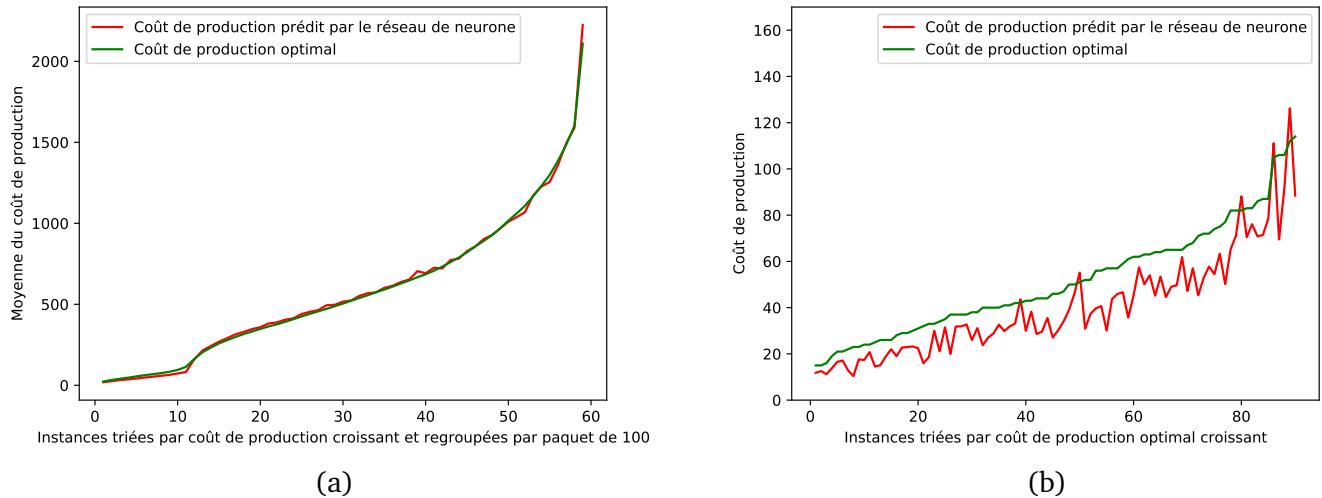


FIGURE 7.21 – Résultats du réseau de neurones **INDIC_COUT** pour la prédiction du coût de production. (a) représente les données d'apprentissage et (b) représente les données de test. La courbe rouge est la courbe des valeurs prédites et la courbe verte est la courbe des valeurs optimales.

7.4.7 Résultats du réseau **MIXTE_TEMPS** pour la prédiction du numéro de période de la dernière recharge

La figure (7.22) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,50] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. On constate que le gap ici se stabilise vite. Au bout d'environ 35 itérations, on a déjà un gap assez stable. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

Les résultats du réseau **MIXTE_TEMPS** pour la prédiction du numéro de période de la dernière recharge sont illustrés par les figures (7.23). On veut voir la corrélation entre les valeurs prédites par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L'axe des abscisses est respectivement pour la figure (7.23) (a) et (7.23) (b) les instances d'apprentissage triées par numéro de période de la dernière recharge croissant et regroupées par paquet de 100 et les instances de test triées par numéro de période de la dernière recharge croissant. L'axe des ordonnées est respectivement pour la figure (7.23) (a) et (7.23) (b) le numéro de période moyen de la dernière opération de recharge et le numéro de période de la dernière opération de recharge. Les points verts sont les valeurs optimales et les points rouges sont les valeurs prédites par le réseau de la figure (7.3). On observe que les valeurs prédites ont l'allure des valeurs optimales.

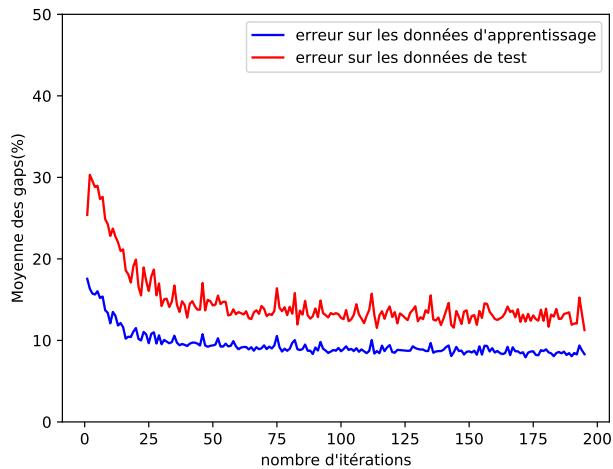


FIGURE 7.22 – Cette figure représente l'évolution du gap lorsqu'on fait varier le nombre d'itérations lors de l'apprentissage du réseau **MIXTE_TEMPS** pour la prédiction du numéro de période de la dernière recharge. La courbe bleue est l'évolution de l'erreur sur les données d'apprentissage et la courbe rouge est l'évolution de l'erreur sur les données de test. Le nombre d'itérations maximal correspond à l'itération qui a fourni le plus petit gap sur les données de test.

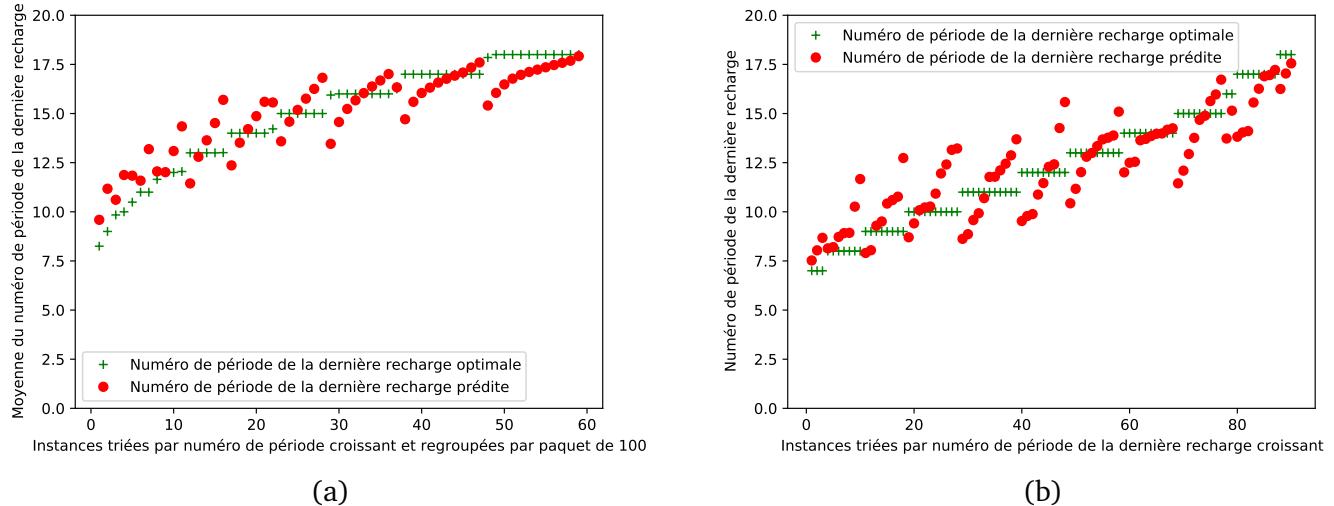


FIGURE 7.23 – Résultats du réseau **MIXTE_TEMPS** pour la prédiction du numéro de période de la dernière recharge. (a) représente les données d'apprentissage et (b) représente les données de test. Les points rouges sont les valeurs prédites et les points verts sont les valeurs optimales.

7.4.8 Résultats du réseau INDIC_TEMPS pour la prédiction du numéro de période de la dernière recharge

La figure (7.24) représente l'évolution du gap moyen des données d'apprentissage (courbe bleue) et des données de test (courbe rouge) lorsqu'on augmente le nombre d'itérations. L'axe des abscisses est le nombre d'itérations et l'axe des ordonnées est le gap moyen. Par souci de lisibilité, on a limité l'axe des ordonnées à l'intervalle [0,50] car on veut pouvoir clairement voir l'écart entre les deux courbes. Globalement, ce gap décroît. On constate que le gap ici se stabilise vite. La troisième itération est celle qui donne la plus petite valeur de gap sur les données de test. Au bout d'environ d'une

itération, on a déjà un gap assez stable. Le nombre d’itérations maximal correspond à l’itération qui a fournit le plus petit gap sur les données de test.

Les résultats du réseau **INDIC_TEMPS** pour la prédiction du numéro de période de la dernière recharge sont illustrés par les figures (7.25). On veut voir la corrélation entre les valeurs prédictes par le réseau et les valeurs optimales utilisées pour entraîner le réseau. L’axe des abscisses est respectivement pour la figure (7.25) (a) et (7.25) (b) les instances d’apprentissage triées par numéro de période de la dernière recharge croissant et regroupées par paquet de 100 et les instances de test triées par numéro de période de la dernière recharge croissant. L’axe des ordonnées est respectivement pour la figure (7.25) (a) et (7.25) (b) le numéro de période moyen de la dernière opération de recharge et le numéro de période de la dernière opération de recharge. Les points verts sont les valeurs optimales et les points rouges sont les valeurs prédictes par le réseau de la figure (7.4). On observe que les valeurs prédictes ont l’allure des valeurs optimales avec une tendance à être au dessus.

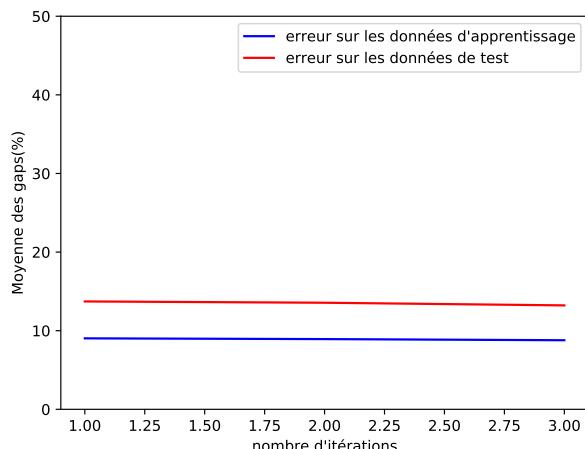


FIGURE 7.24 – Cette figure représente l’évolution du gap lorsqu’on fait varier le nombre d’itérations lors de l’apprentissage du réseau de neurones **INDIC_TEMPS** pour la prédiction du numéro de période de la dernière opération de recharge. La courbe bleue est l’évolution de l’erreur sur les données d’apprentissage et la courbe rouge est l’évolution de l’erreur sur les données de test. Le nombre d’itérations maximal correspond à l’itération qui a fournit le plus petit gap sur les données de test.

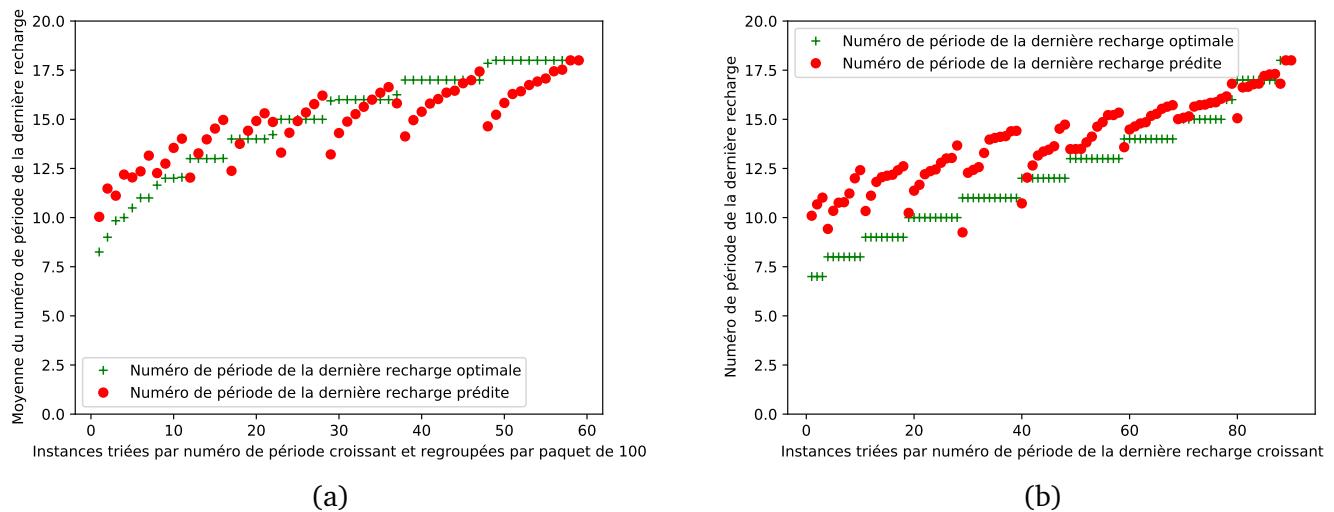


FIGURE 7.25 – Résultats du réseau de neurones **INDIC_TEMPS** pour la prédiction du numéro de période de la dernière opération de recharge. (a) représente les données d'apprentissage et (b) représente les données de test. Les points rouges sont les valeurs prédictives et les points verts sont les valeurs optimales.

7.5 Conclusion

L'objectif de ce chapitre était de chercher à construire des estimateurs du coût de production et du numéro de période de la dernière recharge du problème SMEPC. Ces estimateurs ont été conçus à l'aide de réseaux de neurones. On a présenté 4 réseaux de neurones dont l'objectif était de prédire le coût de production :

- Le réseau **SIMPLE_TYPE** utilise comme entrées des instances classées par type ;
- Le réseau **SIMPLE_PERIODE** utilise comme entrées des instances classées par période ;
- Le réseau **MIXTE_COUT** utilise comme entrées des instances classées par type ;
- Le réseau **INDIC_COUT** utilise comme entrées des indicateurs qui sont une représentation plus compacte des instances.

On a aussi présenté 2 réseaux de neurones dont l'objectif était de prédire le numéro de période de la dernière recharge :

- Le réseau **MIXTE_TEMPS** utilise comme entrées des instances classées par type ;
- Le réseau **INDIC_TEMPS** utilise comme entrées des indicateurs qui sont une représentation plus compacte des instances.

Tout au long de ce chapitre, pour chacun de ces réseaux on a décrit leurs entrées, leur architecture et leurs sorties. On a utilisé pour la phase expérimentale uniquement des instances dont le nombre de périodes est égal à 20. Au cours des expérimentations de nos réseaux, on a tout d'abord généré 6000 instances parmi lesquelles on a utilisé 5910 comme instances d'apprentissage et 90 comme instances de test. Comme critère de performance, on a évalué la qualité des prédictions de nos réseaux de neurones. Après analyse des gaps d'entraînement et des gaps de test, on peut conclure que les réseaux **SIMPLE_TYPE** et **SIMPLE_PERIODE** sont les moins performants.

Les réseaux **INDIC_COUT** et **INDIC_TEMPS** sont les plus performants au vu du faible nombre d'indicateurs utilisés, on a tendance à croire qu'en ajoutant le nombre d'indicateurs ces réseaux auront de meilleures performances.

CHAPITRE 8

CONCLUSION GÉNÉRALE

Sommaire

8.1 Récapitulatif de notre contribution	245
8.2 Perspectives	246

8.1 Récapitulatif de notre contribution

L'objectif de ce manuscrit était de présenter toutes les méthodes qu'on a implémenté pour tenter de résoudre le problème qu'on a nommé SMEPC (voir sa description détaillée à la section 4.2). Le problème est de planifier les recharges d'un véhicule dont on connaît la tournée sachant que les quantités d'énergie rechargées devront préalablement être produite par une machine de production. Dans ce manuscrit, on a présenté plusieurs méthodes de résolution du problème SMEPC. On a principalement six chapitres.

Le **chapitre 2** fait un état de l'art du problème SMEPC. A notre connaissance, il n'existe aucun article qui traite notre problème mais par contre on a trouvé des articles qui traitent certains aspects de notre problème. On a commencé ce chapitre par l'état de l'art des problèmes de planification de tournées de véhicules qui tiennent compte de l'alimentation des véhicules. On a présenté les problèmes GVRP, PRP, EVPRP, HVRP et RVRP. On poursuivi ce chapitre en présentant ce qu'est un problème de planification de la production. On a conclu ce chapitre en identifiant quelques problématique de synchronisation à savoir la synchronisation discrète et la synchronisation continue.

Le **chapitre 3** fait un état de l'art des méthodes explorées pour résoudre le problème SMEPC. On commence par présenter quelques généralités sur la programmation linéaire en nombres entiers et sur le *Branch-And-Cut*. Puis on poursuit en présentant ce qu'est la programmation dynamique en donnant deux exemples de son utilisation sur deux problèmes connus : le problème du sac à dos et le *Split*. On présente comment les problèmes Stochastiques et cycliques peuvent être traités avec cette technique. On finit ce chapitre en présentant ce qu'est l'apprentissage automatique. On se concentre principalement sur les réseaux de neurones et sur l'API Keras de Tensorflow Keras qui sert à les implémenter.

Le **chapitre 4** commence par expliquer le problème SMEPC, ensuite on présente sa formulation mathématique notamment ses variables, sa fonction objectif et ses contraintes. Puis, on donne sa formulation linéaire en nombres entiers qu'on nomme $MILP_{SMEPC}$ puis sa relaxation linéaire qu'on nomme $RMILP_{SMEPC}$. On étudie la contribution de plusieurs contraintes nommées *STC* et *EC* qui améliorent la relaxation linéaire. On étudie la complexité de notre problème et on finit ce chapitre en présentant les résultats des expérimentations numériques effectuées. La phase expérimentale montre que le modèle $MILP_{SMEPC}$ a du mal à résoudre des instances de « grandes » tailles.

Le **chapitre 5** propose un algorithme de programmation dynamique pour tenter de résoudre le problème SMEPC. On nomme cet algorithme DPS_{SMEPC} , on présente l'espace temps, les états, les variables de décisions, les transitions et la mise en œuvre algorithmique de cet algorithme. On propose un schéma d'approximation polynomial. Pour éviter une explosion du nombre d'états, on présente deux types de mécanismes de filtrage : les mécanismes de filtrages exactes (filtrage logique et filtrage par estimation optimiste) et les mécanismes de filtrages heuristiques. On décrit une heuristique rapide nommée *He* qui calcule une solution de notre problème très rapidement. On finit ce chapitre en présentant les résultats des expérimentations numériques effectuées.

Il a été constaté que le problème SMEPC peut être divisé en deux sous-problèmes à savoir le problème *Vehicle-Driver* et le problème *Production-Manager*. Le **chapitre 6** se focalise sur un schéma collaboratif nommé **Pipe-line VD_PM** qui émule une interaction entre ces deux sous-problèmes. On a commencé ce chapitre en présentant ce qu'est le problème VD et le problème PM. Ensuite, on présente un algorithme de programmation dynamique pour chacun de ces problèmes qu'on nomme DPS_VD et DPS_PM . Comme pour le DPS_{SMEPC} , le problème PM étant le plus difficile, on diminue le nombre d'états produit par le DPS_PM à l'aide de mécanismes de filtrage. On finit ce chapitre en

présentant les résultats des multiples expériences effectuées.

Concernant la programmation dynamique, de toutes les expérimentations effectuées tout au long de ce manuscrit, on conclut que le filtrage heuristique est très efficace pour filtrer les états. De plus, le filtrage par estimation optimiste est le filtrage exacte le plus efficace.

Dans le **chapitre 7**, on tente de construire un estimateur rapide du coût de la solution optimale d'une instance. Cet estimateur est construit à l'aide de réseaux de neurones. On commence par construire deux réseaux « simples » qu'on nomme SIMPLE_TYPE et SIMPLE_PERIODE. Ensuite on construit un réseau MIXTE_COUT et MIXTE_TEMPS qui essayent d'épouser les caractéristiques du problème. Les instances étant utilisées comme entrées de nos réseaux, cela a pour conséquence une importante augmentation du nombre de neurones. Pour résoudre ce problème on va construire les réseaux INDIC_COUT et INDIC_TEMPS dont les entrées seront des indicateurs qui sont une agrégation des instances. La phase expérimentale nous montre que les réseaux INDIC_COUT et INDIC_TEMPS sont les plus performants.

8.2 Perspectives

Nous constatons que notre problème peut être étendu, notamment :

- en supposant que l'optimisation de la tournée fait partie du processus de prise de décision ;
- en supposant que les valeurs de rendement R_i sont gérés comme des quantités non déterministes.
- en supposant que le véhicule est alimenté avec plusieurs types d'énergies ;
- en supposant qu'on a plusieurs véhicules pour effectuer les différentes tâches ;

On pourrait concevoir un modèle qui optimise la tournée du véhicule en utilisant le modèle basé sur les techniques d'apprentissage du dernier chapitre. Plus concrètement, On peut évaluer la qualité d'une tournée avec l'algorithme Pipe-line VD_PM dans lequel on va remplacer le module DPS_PM par le réseau de neurones basé sur les indicateurs. Avant cela, on va entraîner le réseau de neurones qui nous permettra de prédire le coût d'une solution de SMEPC. Nous supposons également que nous disposons d'un opérateur de recherche locale qui modifie la tournée en effectuant une séquence d'opérations de suppression, suivie d'une séquence d'opérations de réinsertion de stations.

On pourrait réfléchir comment ajouter de la robustesse à nos algorithmes pour que même si les rendements sont inférieurs aux prévisions que cela n'entraîne pas que les stratégies de production et de recharge soient infaisables. Nous revenons donc ici au problème global SMEPC, et considérons que les rendements de production R_i doivent être gérés comme des quantités non déterministes. Le cadre standard de programmation dynamique stochastique, basé sur des hypothèses markoviennes (Voir section 3.3.3), ne peut être utilisé ici, puisqu'il existe clairement des corrélations entre les R_i . Au lieu d'utiliser donc la programmation dynamique stochastique, on peut s'appuyer sur la notion de **scénario**. Un scénario est une hypothèse de comportement global de l'évolution du R_i , qui peut également être considéré comme une hypothèse sur la façon dont le temps évolue au cours des périodes $0, \dots, N - 1$.

Le problème SMEPC peut être étendu en modifiant ses caractéristiques. Si on décide que le véhicule fonctionne avec deux types d'énergies, par exemple l'hydrogène et l'électricité. Ceci signifie que le véhicule contient une pile à hydrogène pour conserver l'hydrogène et une batterie pour stocker l'électricité. L'une des difficultés ici serait de calculer quelle proportion d'hydrogène et d'électricité le

véhicule devrait dépenser pour se déplacer d'une station à l'autre. De plus, il faudrait décider quelles proportions d'hydrogène et d'électricité seront rechargée lorsque le véhicule ira se recharger.

Si on décide qu'il y a plusieurs véhicules (au lieu d'un seul) pour effectuer les tâches de logistique interne, on a plusieurs difficultés : l'une est d'empêcher les collisions entre les véhicules en faisant en sorte qu'ils ne se croisent jamais sur la même route ou à un carrefour. Une autre difficulté est d'attribuer des tâches de façon optimale à chaque véhicule. Aussi, on doit gérer une file d'attente au niveau de la micro-usine car les véhicules peuvent s'y retrouver à plusieurs. Une autre difficulté est d'optimiser les tournées des véhicules.

Le paragraphe suivant sera consacré à la présentation plus ou moins exhaustive de façon précise des extensions possibles du problème **SMEPC**. Parmi les variants du problème **SMEPC** avec tournée fixée, on peut citer :

1. On suppose que la quantité d'hydrogène rechargée par le véhicule est la même à chaque recharge, la recharge dure une période. La quantité d'hydrogène produite est variable (La micro-usine produit à chaque pas de temps une quantité d'hydrogène comprise entre 1 et un seuil max à fixer.). Le véhicule n'attend pas au niveau de la micro-usine, il se recharge immédiatement car il est prioritaire ;
2. On suppose que la quantité d'hydrogène rechargée par le véhicule est la même à chaque recharge, la recharge dure une période. La quantité d'hydrogène produite est fixe (La micro-usine produit à chaque pas de temps une quantité d'hydrogène connue.). Le véhicule n'attend pas au niveau de la micro-usine, il se recharge immédiatement car il est prioritaire ;
3. On suppose que la quantité d'hydrogène rechargée par le véhicule est la même à chaque recharge, la recharge se fait en δ périodes de temps. La quantité d'hydrogène produite est fixe et on n'a pas de temps d'attente ;
4. Le véhicule fait le plein de son réservoir d'hydrogène à chaque recharge, la recharge se fait en δ périodes de temps, la quantité d'hydrogène produite est fixe et on n'a pas de temps d'attente ;
5. Le véhicule fait le plein de son réservoir d'hydrogène à chaque recharge, la recharge dure une période, la quantité d'hydrogène produite est fixe et le véhicule peut attendre à la micro-usine (par exemple il peut attendre que la micro-usine produise la quantité dont il a besoin). Ce dernier cas est le problème **SMEPC**.

Le tableau (8.1) synthétise les variants du problème **SMEPC** présenté ci-dessus.

Caractéristiques	Possibilités					SMEPC
Quantité rechargée	fixe	1	2	3	4	5
	Variable					
Durée de la recharge	δ périodes			3	4	5
	1 période	1	2			
Quantité produite	fixe		2	3	4	5
	variable	1				
Attente avant recharge	oui					5
	non	1	2	3	4	

TABLE 8.1 – Synthèse de quelques variants du problème **SMEPC**.

BIBLIOGRAPHIE

- [1] Samir A. Abass, Mohamed Ali Gomaa, Gaber A. Elsharawy, and Marwa Sh. Elsaied. Generalized production planning problem under interval uncertainty. *Egyptian Informatics Journal*, 11(1) :27 – 31, 2010.
- [2] Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1) :103–120, 2014.
- [3] Reza Alizadeh Foroutan, Javad Rezaeian, and Iraj Mahdavi. Green vehicle routing and scheduling problem with heterogeneous fleet including reverse logistics in the form of collecting returned goods. *Applied Soft Computing*, 94 :106462, 2020.
- [4] Aldair Alvarez, Jean-François Cordeau, Raf Jans, Pedro Munari, and Reinaldo Morabito. Formulations, branch-and-cut and a hybrid heuristic algorithm for an inventory routing problem with perishable products. *European Journal of Operational Research*, 283, 11 2019.
- [5] Aldair Alvarez, Pedro Munari, and Reinaldo Morabito. Iterated local search and simulated annealing algorithms for the inventory routing problem. *International Transactions in Operational Research*, 25(6) :1785–1809, 2018.
- [6] Juho Andelmin and Enrico Bartolini. An exact algorithm for the green vehicle routing problem. *Transportation Science*, 51, 07 2017.
- [7] Eric Angel, Evripidis Bampis, and Vincent Chau. Low complexity scheduling algorithms minimizing the energy for tasks with agreeable deadlines. *Discrete Applied Mathematics*, 175 :1 – 10, 2014.
- [8] C. Archetti, Luca Bertazzi, Alain Hertz, and M.Grazia Speranza. A hybrid algorithm for an inventory-routing problem. *Informs Journal on Computing*, 24 :101–116, 01 2012.
- [9] C. Archetti, Luca Bertazzi, Gilbert Laporte, and M.Grazia Speranza. A branch-and-cut algorithm for a vendor-managed inventory-routing problem. *Transportation Science*, 41 :382–391, 08 2007.
- [10] Claudia Archetti, Natashia Boland, and M. Grazia Speranza. A matheuristic for the multivehicle inventory routing problem. *INFORMS Journal on Computing*, 29(3) :377–387, 2017.
- [11] Dmitry I. Arkhipov, Olga Battaïa, and Alexander A. Lazarev. Long-term production planning problem : scheduling, makespan estimation and bottleneck analysis. *IFAC-PapersOnLine*, 50(1) :7970 – 7974, 2017. 20th IFAC World Congress.

- [12] Sina Bahrami, Mehdi Nourinejad, Glareh Amirjamshidi, and Matthew J. Roorda. The plugin hybrid electric vehicle routing problem : A power-management strategy model. *Transportation Research Part C : Emerging Technologies*, 111 :318 – 333, 2020.
- [13] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods : A polynomial time algorithm for offline dynamic power management. pages 364–367, 01 2006.
- [14] Tolga Bektaş and Gilbert Laporte. The pollution-routing problem. *Transportation Research Part B : Methodological*, 45(8) :1232–1250, 2011. Supply chain disruption and risk management.
- [15] Richard Bellman. Dynamic programming. *Science*, 153(3731) :34–37, 1966.
- [16] Pascale Bendotti, Philippe Chrétienne, Pierre Fouilhoux, and Alain Quilliot. Anchored reactive and proactive solutions to the CPM-scheduling problem. *European Journal of Operational Research*, 261(1) :67–74, 2017.
- [17] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3) :299–316, 2000.
- [18] Barry W Brown. On the iterative method of dynamic programming on a finite space discrete time markov process. *The annals of mathematical statistics*, pages 1279–1285, 1965.
- [19] Andrew Burke. Batteries and ultracapacitors for electric, hybrid, and fuel cell vehicles. *Proceedings of the IEEE*, 95 :806 – 820, 05 2007.
- [20] C.C. Chan. The state of the art of electric, hybrid, and fuel cell vehicles. *Proceedings of the IEEE*, 95 :704 – 718, 05 2007.
- [21] Sandeep Singh Chauhan, Chinta Sivadurgaprasad, Rajasekhar Kadambur, and Prakash Kotecha. A novel strategy for the combinatorial production planning problem using integer variables and performance evaluation of recent optimization algorithms. *Swarm and Evolutionary Computation*, 43 :225 – 243, 2018.
- [22] Rui Chen, Xinwu Qian, Lixin Miao, and Satish V. Ukkusuri. Optimal charging facility location and capacity for electric vehicles considering route choice and charging time equilibrium. *Computers and Operations Research*, 113 :104776, 2020.
- [23] Masoud Chitsaz, Jean-François Cordeau, and Raf Jans. A unified decomposition matheuristic for assembly, production, and inventory routing. *INFORMS Journal on Computing*, 31(1) :134–152, 2019.
- [24] N. Christofides, A. Mingozzi, and P. Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2) :145–164, 1981.
- [25] Leandro C. Coelho, Jean-François Cordeau, and Gilbert Laporte. Consistency in multi-vehicle inventory-routing. *Transportation Research Part C-emerging Technologies*, 24 :270–287, 2012.
- [26] Leandro C Coelho and Gilbert Laporte. Improved solutions for inventory-routing problems through valid inequalities and input ordering. *International Journal of Production Economics*, 155(C) :391–397, 2014.
- [27] Weiwei Cui, Zhiqiang Lu, Chen Li, and Xiaole Han. A proactive approach to solve integrated production scheduling and maintenance planning problem in flow shops. *Computers and Industrial Engineering*, 115 :342 – 353, 2018.

- [28] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1) :80–91, 1959.
- [29] Erik Demaine, Mohammad Ghodsi, Mohammad Hajiaghayi, Amin Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *Journal of Scheduling*, 16 :46–54, 06 2007.
- [30] E. Demir, T. Bektas, and G. Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2) :346–359, December 2012.
- [31] E. Demir, T. Bektas, and G. Laporte. The bi-objective pollution-routing problem. *European Journal of Operational Research*, 232(3) :464–478, 2013.
- [32] C. Doga Demirhan, Fani Boukouvala, Kyungwon Kim, Hyeju Song, William W. Tso, Christodoulos A. Floudas, and Efstratios N. Pistikopoulos. An integrated data-driven modeling and global optimization approach for multi-period nonlinear production planning problems. *Computers and Chemical Engineering*, 141 :107007, 2020.
- [33] Guy Desaulniers, Jørgen G. Rakke, and Leandro C. Coelho. A Branch-Price-and-Cut Algorithm for the Inventory-Routing Problem. *Transportation Science*, 50(3) :1060–1076, August 2016.
- [34] M.V. Devyaterikova, A.A. Kolokolov, and A.P. Kolosov. L-class enumeration algorithms for a discrete production planning problem with interval resource quantities. *Computers and Operations Research*, 36(2) :316 – 324, 2009. Scheduling for Modern Manufacturing, Logistics, and Supply Chains.
- [35] Michael Drexel. Synchronization in Vehicle Routing—A Survey of VRPs with Multiple Synchronization Constraints. *Transportation Science*, 46(3) :297–316, August 2012.
- [36] Sevgi Erdoğan and Elise Miller-Hooks. A green vehicle routing problem. *Transportation Research Part E : Logistics and Transportation Review*, 109 :100–114, 01 2012.
- [37] Saman Eskandarzadeh, Kourosh Eshghi, and Mohsen Bahramgiri. Risk shaping in production planning problem with pricing under random yield. *European Journal of Operational Research*, 253(1) :108 – 120, 2016.
- [38] S. Rasoul Etesami, Walid Saad, Narayan B. Mandayam, and H. Vincent Poor. Smart routing of electric vehicles for load balancing in smart grids. *Automatica*, 120 :109148, 2020.
- [39] Martin Fink, Guy Desaulniers, Markus Frey, Ferdinand Kiermaier, Rainer Kolisch, and François Soumis. Column generation for vehicle routing problems with multiple synchronization constraints. *European Journal of Operational Research*, 272(2) :699 – 711, 2019.
- [40] Anna Franceschetti, Emrah Demir, Dorothée Honhon, Tom Van Woensel, Gilbert Laporte, and Mark Stobbe. A metaheuristic for the time-dependent pollution-routing problem. *European Journal of Operational Research*, 259(3) :972 – 991, 2017.
- [41] Anna Franceschetti, Dorothée Honhon, Tom Van Woensel, Tolga Bektaş, and Gilbert Laporte. The time-dependent pollution-routing problem. *Transportation Research Part B : Methodological*, 56(C) :265–293, 2013.
- [42] Yuvraj Gajpal and Mustapha Noureldath. Two efficient heuristics to solve the integrated load distribution and production planning problem. *Reliability Engineering and System Safety*, 144 :204 – 214, 2015.

- [43] Antonio Giallanza and Gabriella Li Puma. Fuzzy green vehicle routing problem for designing a three echelons supply chain. *Journal of Cleaner Production*, 259 :120774, 2020.
- [44] Craig Grimes, Oomman Varghese, and Sudhir Ranjan. *Light, water, hydrogen : The solar generation of hydrogen by water photoelectrolysis*. 01 2008.
- [45] Yun He, Cyril Briand, and Nicolas Jozefowicz. A mass-flow based milp formulation for the inventory routing with explicit energy consumption. pages 242–251, 02 2016.
- [46] Fanny Hein and Christian Almeder. Quantitative insights into the integrated supply vehicle routing and production planning problem. *International Journal of Production Economics*, 177 :66 – 76, 2016.
- [47] Mike Hewitt, George Nemhauser, Martin Savelsbergh, and Jin-Hwa Song. A branch-and-price guided search approach to maritime inventory routing. *Computers and Operations Research*, 40 :1410–1419, 05 2013.
- [48] Labex IMobS3. Plateformes Labex IMobS3. <http://www.imobs3.uca.fr/index.php/fr/menu-valorisation/plateformes>.
- [49] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *sigact news*, 36(2), 63–76. *SIGACT News*, 36 :63–76, 06 2005.
- [50] Surendra Reddy Kancharla and Gitakrishnan Ramadurai. Electric vehicle routing problem with non-linear charging and load-dependent discharging. *Expert Systems with Applications*, 160 :113714, 2020.
- [51] İmdat Kara, Bahar Y. Kara, and M. Kadri Yetis. Energy minimizing vehicle routing problem. In Andreas Dress, Yinfeng Xu, and Binhai Zhu, editors, *Combinatorial Optimization and Applications*, pages 62–71, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [52] Panagiotis Karakostas, Angelo Sifaleras, and Michael C. Georgiadis. Adaptive variable neighborhood search solution methods for the fleet size and mix pollution location-inventory-routing problem. *Expert Systems with Applications*, 153 :113444, 2020.
- [53] Richard M Karp and Michael Held. Finite-state processes and dynamic programming. *SIAM Journal on Applied Mathematics*, 15(3) :693–718, 1967.
- [54] Çağrı Koç, Ola Jabali, Jorge Mendoza, and Gilbert Laporte. The electric vehicle routing problem with shared charging stations. *International Transactions in Operational Research*, 26, 11 2018.
- [55] Çağrı Koç and Ismail Karaoglan. The green vehicle routing problem : A heuristic based exact solution approach. *Applied Soft Computing*, 39 :154–164, 03 2016.
- [56] Çağrı Koç, Tolga Bektaş, Ola Jabali, and Gilbert Laporte. The fleet size and mix pollution-routing problem. *Transportation Research Part B : Methodological*, 70 :239–254, December 2014.
- [57] A. W. J. Kolen, A. H. G. Rinnooy Kan, and H. W. J. M. Trienekens. Vehicle routing with time windows. *Operations Research*, 35(2) :266–273, 1987.
- [58] R. Küttner and J. Majak. Multistage optimisation strategy for solving production planning problems. *IFAC Proceedings Volumes*, 42(2) :332 – 337, 2009. 14th IFAC Workshop on Control Applications of Optimization.
- [59] Yiyo Kuo. Using simulated annealing to minimize fuel consumption for the time-dependent vehicle routing problem. *Computers and Industrial Engineering*, 59 :157–165, 08 2010.

- [60] Kexing Lai, Tao Chen, and Balasubramaniam Natarajan. Optimal scheduling of electric vehicles car-sharing service with multi-temporal and multi-task operation. *Energy*, 204 :117929, 2020.
- [61] Antti Lajunen. Energy consumption and cost-benefit analysis of hybrid and electric city buses. *Transportation Research Part C : Emerging Technologies*, 38 :1–15, 01 2014.
- [62] Yan-Fei Lan, Yan-Kui Liu, and Gao-Ji Sun. Modeling fuzzy multi-period production planning and sourcing problem with credibility service levels. *Journal of Computational and Applied Mathematics*, 231(1) :208 – 221, 2009.
- [63] Yanfei Lan, Yankui Liu, and Gaoji Sun. An approximation-based approach for fuzzy multi-period production planning problem with credibility objective. *Applied Mathematical Modelling*, 34(11) :3202 – 3215, 2010.
- [64] Jingran Liang, Yuyan Wang, Zhi-Hai Zhang, and Yiqi Sun. Energy efficient production planning and scheduling problem with processing technology selection. *Computers and Industrial Engineering*, 132 :260 – 270, 2019.
- [65] Stuart Licht. *Thermochemical and Thermal/Photo Hybrid Solar Water Splitting*, pages 87–121. Springer New York, New York, NY, 2008.
- [66] Canhong Lin, K.L. Choy, G.T.s Ho, Sai-Ho Chung, and H. Lam. Survey of green vehicle routing problem : Past and future trends. *Expert Systems with Applications*, 41 :1118–1138, 03 2014.
- [67] Ran LIU, Yangyi TAO, and Xiaolei Xie. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers and Operations Research*, 101, 08 2018.
- [68] Ji Lu, Yunling Chen, Jin-Kao Hao, and Renjie He. The time-dependent electric vehicle routing problem : Model and solution. *Expert Systems with Applications*, 161 :113593, 2020.
- [69] Simona Mancini. The hybrid vehicle routing problem. *Transportation Research Part C : Emerging Technologies*, 78 :1–12, 05 2017.
- [70] Marcos Raylan Sousa Matos, Yuri Frota, and Luiz Satoru Ochi. Green vehicle routing and scheduling problem with split delivery. *Electronic Notes in Discrete Mathematics*, 69 :13 – 20, 2018. Joint EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization (EURO/ALIO 2018).
- [71] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5 :127–147, 1943.
- [72] Esmaeil Mehdizadeh, Seyed Taghi Akhavan Niaki, and Mojtaba Hemati. A bi-objective aggregate production planning problem with learning effect and machine deterioration : Modeling and solution. *Computers and Operations Research*, 91 :21 – 36, 2018.
- [73] LG Mitten. Composition principles for synthesis of optimal multistage processes. *Operations Research*, 12(4) :610–619, 1964.
- [74] Joon-Yung Moon and Jin-Woo Park. Smart production scheduling with time-dependent and machine-dependent electricity cost by considering distributed energy resources and energy storage. *International Journal of Production Research*, 52 :18, 12 2013.
- [75] Joon-Yung Moon, Kitae Shin, and Jinwoo Park. Optimization of production scheduling with time-dependent and machine-dependent electricity cost for industrial energy efficiency. *The International Journal of Advanced Manufacturing Technology*, 68(1) :523–535, Sep 2013.

- [76] Haouassi Mustapha, Chloé Desdouits, Rodolphe Giroudeau, and Claude Pape. Production scheduling with a piecewise-linear energy cost function. pages 1–8, 12 2016.
- [77] George L Nemhauser. Introduction to dynamic programming. 1966.
- [78] Nur Normasari, Vincent Yu, Candra Bachtiyar, and Sukoyo. A simulated annealing heuristic for the capacitated green vehicle routing problem. *Mathematical Problems in Engineering*, 2019 :1–18, 01 2019.
- [79] F.Sourd P.Baptiste, E.Neron. Modèles et algorithmes en ordonnancement. pages 198–203, 2004.
- [80] Agnes Pechmann and Ilka Schöler. Optimizing energy costs by intelligent production scheduling. In Jürgen Hesselbach and Christoph Herrmann, editors, *Glocalized Solutions for Sustainability in Manufacturing*, pages 293–298, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [81] Rui Qiu, Jiuping Xu, Ruimin Ke, Ziqiang Zeng, and Yinhai Wang. Carbon pricing initiatives-based bi-level pollution routing problem. *European Journal of Operational Research*, 286(1) :203 – 217, 2020.
- [82] Alain Quilliot and Philippe Chrétienne. Homogeneously non-idling schedules of unit-time jobs on identical parallel machines. *Discrete Applied Mathematics*, 161 :1586–1597, July 2013.
- [83] Alain Quilliot and Philippe Chrétienne. Homogeneously non-idling schedules of unit-time jobs on identical parallel machines. *Discrete Applied Mathematics*, 161 :1586–1597, 07 2013.
- [84] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40 :455–472, 11 2006.
- [85] Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient energy-optimal routing for electric vehicles. In *AAAI*, 2011.
- [86] M. Sakarovitch. *Optimisation combinatoire, programmation discrète*, volume 2. Hermann, 1984.
- [87] Edcarlos Santos, Luiz Satoru Ochi, Luidi Simonetti, and Pedro Henrique González. A hybrid heuristic based on iterated local search for multivehicle inventory routing problem. *Electronic Notes in Discrete Mathematics*, 52 :197 – 204, 2016. INOC 2015 – 7th International Network Optimization Conference.
- [88] Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48 :500–520, 03 2014.
- [89] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. 09 1997.
- [90] Albers Susanne. Energy-efficient algorithms. *Commun. ACM*, 53 :86–96, 05 2010.
- [91] Lixin Tang, Ping Che, and Jiyin Liu. A stochastic production planning problem with nonlinear cost. *Computers and Operations Research*, 39(9) :1977 – 1987, 2012.
- [92] Erfan Babaee Tirkolaee, Alireza Goli, Amin Faridnia, Mehdi Soltani, and Gerhard-Wilhelm Weber. Multi-objective optimization for the reliable pollution-routing problem with cross-dock selection using pareto-based algorithms. *Journal of Cleaner Production*, 276 :122927, 2020.
- [93] Nicolás Vanzetti, Diego Broz, Jorge M. Montagna, and Gabriela Corsano. Integrated approach for the bucking and production planning problems in forest industry. *Computers and Chemical Engineering*, 125 :155 – 163, 2019.

- [94] Rashid Waraich, Matthias Galus, Christoph Dobler, Michael Balmer, Göran Andersson, and Kay Axhausen. Plug-in hybrid electric vehicles and smart grid : Investigations based on a micro simulation. *Transportation Research Part C Emerging Technologies*, 28, 06 2014.
- [95] Douglass J Wilde and Charles S Beightler. Foundations of optimization. 1967.
- [96] Yiyong Xiao and Abdullah Konak. The heterogeneous green vehicle routing and scheduling problem with time-varying traffic congestion. *Transportation Research Part E : Logistics and Transportation Review*, 88(C) :146–166, 2016.
- [97] Yiyong Xiao, Xiaorong Zuo, Jiaoying Huang, Abdullah Konak, and Yuchun Xu. The continuous pollution routing problem. *Applied Mathematics and Computation*, page 125072, 2020.
- [98] Yiyong Xiao, Xiaorong Zuo, Ikou Kaku, Shenghan Zhou, and Xing Pan. Development of energy consumption optimization model for the electric vehicle routing problem with time windows. *Journal of Cleaner Production*, 225, 04 2019.
- [99] Guisen Xue and O. Felix Offodile. Integrated optimization of dynamic cell formation and hierarchical production planning problems. *Computers and Industrial Engineering*, 139 :106155, 2020.
- [100] Vincent Yu, Perwira Redi, Yosi Agustina, and Oktaviyanto Wibowo. A simulated annealing heuristic for the hybrid vehicle routing problem. *Applied Soft Computing*, 53, 12 2016.
- [101] Shuai Zhang, Mingzhou Chen, Wenyu Zhang, and Xiaoyu Zhuang. Fuzzy optimization model for electric vehicle routing problem with time windows and recharging stations. *Expert Systems with Applications*, 145 :113123, 2020.
- [102] Shuai Zhang, Yuvraj Gajpal, S. Appadoo, and Mohamed Abdulkader. Electric vehicle routing problem with recharging stations for minimizing energy consumption. *International Journal of Production Economics*, 203, 07 2018.
- [103] Hui Zhao, Edward Huang, Runliang Dou, and Kan Wu. A multi-objective production planning problem with the consideration of time and cost in clinical trials. *Expert Systems with Applications*, 124 :25 – 38, 2019.
- [104] Lu Zhen, Ziheng Xu, Chengle Ma, and Liyang Xiao. Hybrid electric vehicle routing problem with mode selection. *International Journal of Production Research*, pages 1–15, 03 2019.
- [105] Ferani E. Zulvia, R.J. Kuo, and Dwiyanti Y. Nugroho. A many-objective gradient evolution algorithm for solving a green vehicle routing problem with time windows and time dependency for perishable products. *Journal of Cleaner Production*, 242 :118428, 2020.