Modélisation par contraintes et heuristiques pour l'évaluation de la capacité d'infrastructures ferroviaires

Fabien Degoutin

Université de Valenciennes et du Hainaut-Cambrésis

Numéro d'ordre : 0754

THÈSE

pour l'obtention du

Doctorat de l'université de Valenciennes et du Hainaut-Cambrésis

Spécialité Automatique et Informatique des Systèmes Industriels et Humains

Discipline : Informatique

Modélisation par contraintes et heuristiques pour l'évaluation de la capacité d'infrastructures ferroviaires

présentée par

Fabien DEGOUTIN

Soutenue publiquement le 12 décembre 2007 devant le jury composé de :

Michel VASQUEZ	Maître Assistant, HDR, École des Mines d'Alès	Rapporteur
Philippe LACOMME	Maître de Conférence, HDR, Université Blaise Pascal, Clermont-Ferrand	Rapporteur
David DE ALMEIDA	Chercheur, Direction de l'Innovation et de la Recherche de la SNCF, Paris	Invité
Fabien LE HUÉDÉ	Chargé de Recherches, École des Mines de Nantes	Examinateur
Saïd HANAFI	Professeur, Université de Valenciennes et du Hainaut-Cambrésis	Examinateur
Arnaud FRÉVILLE	Professeur, Conseil Régional Nord-Pas-de-Calais	Directeur
Joaquín RODRIGUEZ	Directeur de Recherche, INRETS, Villeneuve D'Ascq	Co-directeur
Xavier GANDIBLEUX	Professeur, Université de Nantes	Co-directeur

Avant-propos

Mes journées de doctorat n'ont pas toujours été roses, mais grâce à l'encadrement et l'entourage que j'ai pu avoir, cela fut plus aisé.

Je pense notamment à mes responsables de thèse Messieurs Arnaud Fréville, Professeur et Directeur de la recherche au conseil régional du Nord-Pas-De-Calais, Joaquin Rodriguez, Directeur de Recherche à l'Institut National de Recherche sur les Transports et leur Sécurité (INRETS) et Xavier Gandibleux, Professeur à l'Université de Nantes. Je tiens à les remercier pour le temps qu'ils ont pu m'accorder et pour leur patience.

Messieurs les membres du jury, ils ont pris le temps de lire, évaluer et commenter mon travail. Ma gratitude va donc vers Messieurs Michel Vasquez, Maître Assistant à l'École des Mines d'Alès, Philippe LACOMME, Maître de Conférence et HDR de l'Université Blaise Pascal de Clermont-Ferrand, David De Almeida, Chercheur à la SNCF, Fabien Le Huédé, Chargé de Recherches et Saïd HANAFI, Professeur de l'Université de Valenciennes et du Hainaut-Cambrésis.

J'ai pu bénéficier, de nombreuses fois, de l'expérience de Xavier Delorme, Maître assistant à l'École des Mines de Saint Étienne. Nos nombreuses discussions, professionelle ou non, m'ont toujours été très utiles, merci à toi Xavier.

Les équipes qui m'ont accueilli et aidé durant mes travaux est l'équipe ESTAS de l'INRETS Villeneuve d'Ascq, dirigée à mon arrivée par Gérard Couvreur et à mon départ par El-Miloudi El-Koursi, ainsi que l'équipe ROI de l'Université de Valenciennes et du Hainaut-Cambrésis dirigée par Frédéric Semet où j'ai étudié et où j'ai eu la chance d'enseigner. Ils m'ont tous soutenus dans mes travaux, je pense notamment à Grégory Marlière, Philippe Bon (s13), Georges Mariano et Sonia Sobieraj, Luce Brotcorne, Ioana-Codrutsa Bilegan, Jean Marie Raviart, merci à eux.

J'adresse également tous mes remerciements aux personnes que j'ai cotoyé, je tiens à saluer mes amis Othman Lahlou, Mohamed-Amine Kamoun et Jérôme Rocheteau, ainsi que Sana, Meriem, Joffrey et François.

Enfin je remercie ma relectrice et professeur d'orthographe privilégiée et préférée ainsi que ma famille pour leur gentillesse et les moments de joie et de détente qu'ils ont pu me donner.

Table des matières

Table des figures

Liste des tableaux

Introduction

Ce mémoire présente les travaux effectués lors de ma thèse à l'INRETS (Institut National de la Recherche sur les Transports et leur Sécurité) au sein de l'unité de recherche ESTAS (Evaluation des Systèmes de Transports Automatisés et de leur Sécurité) ainsi qu'au LAMIH (Laboratoire d'Automatique, de Mécanique et d'informatique Industrielles et Humaines) équipe ROI (Recherche Opérationnelle et Informatique) de l'Université de Valenciennes et du Hainaut Cambrésis. Ces travaux s'inscrivent dans la thématique de la recherche opérationnelle et de la programmation par contraintes, son domaine d'application est la planification ferroviaire.

Par rapport aux autres moyens de transport, le transport ferroviaire présente de nombreux avantages du point de vue du développement durable. Cependant, dans un environnement économique de plus en plus concurrentiel, ces avantages ne sont pas suffisants pour lui permettre de s'imposer. Ainsi, la SNCF (Société Nationale des Chemins de fer Français) se doit d'améliorer l'efficacité du transport ferroviaire et la qualité des services aux clients. La ponctualité, la fréquence des trains, les coûts et la sécurité sont les critères prépondérants. La qualité de la planification joue un rôle central pour l'optimisation de ces critères.

L'établissement d'un plan de transport est un art difficile et délicat. En raison des contraintes fortes liées à l'exploitation ferroviaire et au maillage important du réseau, un travail de conception et de planification est réalisé en amont, pour l'ensemble des ressources à utiliser sur un an (infrastructure, locomotives, agents). Ce travail de conception est suivi d'une phase d'adaptation, de quelques jours à quelques semaines avant l'éxecution effective du plan de transport. Les adaptations portent sur les roulements des engins, des agents, pour tenir compte, par exemple de modifications d'horaires, de trains supplémentaires ... En phase opérationnelle les agents en charge de l'affectation des locomotives et des personnels aux marches des trains partent du résultat issu du travail préliminaire de conception et d'adaptation. Dans cette phase, une série d'aléas (retards, pannes, ...) peut entraîner des incompatibilités d'affectation que les agents doivent traiter. Pour résoudre les incompatibilités, les agents ré-affectent certaines resources tout en essayant de réduire l'impact de ces modifications sur l'ensemble du plan.

Comme nous venons de le voir, une phase amont conséquente est à effectuer afin de concevoir le plan de transport cohérent et le plus robuste possible. C'est dans cette phase que l'offre de circulation est établie. Il est alors primordial de connaître précisément la capacité des infrastructures

utilisées, car cela permet, notamment :

- d'évaluer la faisabilité d'une offre,
- de localiser les points bloquants.

Si une offre n'est pas faisable, il faudra alors soit en proposer une autre, soit entreprendre des travaux d'améliorations des infrastructures aux niveaux des points bloquants. L'évaluation de la capacité d'infrastructures ferroviaires prend alors toute sa dimension.

Ainsi l'étude de la capacité d'infrastructure ferroviaire permet :

- d'estimer les limites d'un réseau par rapport à une offre et ainsi de mieux envisager les évolutions d'une infrastructure pour faire face aux évolutions de la demande,
- d'aider à la conception d'une offre future à partir d'une estimation de l'évolution de la demande, en utilisant au mieux l'existant,
- faciliter l'étude de modifications d'infrastructure en évaluant l'intérêt (par exemple : le nombre de trains supplémentaires pouvant circuler) de variantes de projets.

Cette évaluation est d'autant plus importante, que la demande est appelée à évoluer et à croître fortement. Ces modifications et croissance sont dues notamment à la concurrence des autres modes de transport, des besoins toujours plus importants en terme de mobilité et du fait que le transport ferroviaire est un bon paliatif à la route.

Ces études de capacité peuvent être réalisées relativement simplement pour des infrastructures simples, mais pour des infrastructures plus complexes, l'emploi de méthodes algorithmiques est nécessaire. En effet, l'étude d'une portion importante du réseau ou d'une portion comportant de nombreux croisements (tel un nœud ou une gare) est plus complexe que l'étude de voies simples. L'accroissement des circulations transforme alors certains nœuds en véritable goulets d'étranglement, ceci rendant très critique la gestion du trafic. Ainsi ces nœuds se comportent, dans ce cas, comme des amplificateurs de retards, car tout retard d'un train pourra avoir un impact important sur ceux empruntant le même nœud.

Ce type de problème s'avère complexe à résoudre, c'est pourquoi des outils mathématiques provenant de la recherche opérationnelle et de la programmation par contraintes ont été proposés. Ces outils permettent d'élaborer des algorithmes capables de répondre de manière efficace aux problèmes complexes. L'objectif du travail effectué a consisté à la proposition d'une modélisation du problème de l'évaluation de la capacité ferroviaire, à l'échelle d'un nœud, ainsi qu'à l'élaboration des algorithmes pour le résoudre.

Le modèle et la résolution proposés pour le problème d'évaluation de la capacité d'infrastructures ferroviaires s'appuient aussi bien sur les techniques de programmation par contraintes que celles de la recherche opérationnelle (ces deux techniques étant très étroitements liées). Le modèle proposé permet de tenir compte de manière très précise des caractéristiques de l'infrastructure, notamment de son système de signalisation et des différents types de trains pouvant circuler. Une série d'améliorations sur ce modèle est apportée. Elles portent sur la réduction du nombre de contraintes

et variables, la ré-écriture de contraintes, l'ajout de coupes et bornes ainsi que sur la suppression des symétries (solutions équivalentes). Ces améliorations ont pu être obtenues notamment grâce aux spécificités du problème de capacité. Un algorithme de recherche exacte a été développé, mais aux vues des temps de traitement prohibitifs, l'étude de métaheuristiques a été nécessaire. La métaheuristique implémentée s'inspire de la métaheuristique de recherche à voisinage large (LNS), proposée par Shaw [Shaw98], ainsi que de la métaheuristique à voisinage variable dont une présentation a été faite par Mladenovic [Mladenovic et al.97] et enfin de l'heuristique de recherche à arborescence tronquée DDS proposée par Walsh [Walsh97]. Des expérimentations menées sur des instances réelles du nœud de Pierrefitte-Gonnesse, et des instances aléatoires, permettent de démontrer l'intérêt des diverses améliorations, ainsi que de montrer l'efficacité de la métaheuristique développée sur des tailles d'instances répondant aux évaluations souhaitées et concrètes, mais aussi en des temps de résolution raisonnables.

Ce mémoire est divisée en plusieurs chapitres, dont voici une courte présentation :

Le chapitre 1 présente la problématique de ce travail. Une rapide présentation de l'infrastructure ferroviaire et des différents types de conflits entre trains est effectuée. L'intérêt d'évaluer la capacité d'une infrastructure y est décrit. Suite à quoi la problématique de la saturation d'une infrastructure ferroviaire est posée et définie. Une étude bibliographique des travaux existants est proposée. Le positionnement et la singularité de ce travail au sein de ces différentes études sont montrés.

Le second chapitre fournit les éléments utilisés pour la modélisation et la résolution de notre problématique. La programmation par contraintes et l'ensemble de ces techniques de modélisation (problème de satisfaction de contraintes ou CSP) et de propagation sont ainsi définis et expliqués. De même, la problématique des solutions symétriques est posée. Les CSP temporels permettant de représenter des problèmes temporels et possédant des algorithmes de filtrages appropriés sont décrits. Les différents algorithmes de résolution exacte et approchée que nous utiliserons sont ici présentés et détaillés.

La modélisation utilisée pour représenter notre problème fait l'objet du troisième chapitre. Notre modélisation se base sur les conflits de circulation, aussi une première partie présente la gestion des conflits de circulation. Elle en donne une définition et présente les hypothèses de notre étude. Puis le modèle de base est décrit, il est composé de deux types de variables (variables d'affectations de ressources et d'ordonnancement de trains), des contraintes d'énumération et d'incompatibilité et d'un critère. Une série d'améliorations de ce modèle est alors proposée. Enfin la représentation des infrastructures, type de trains et parcours est faite.

Les algorithmes de résolution utilisés, les améliorations proposées et leurs différents paramètres sont exposés au quatrième chapitre. La résolution prend en compte la stratégie d'énumération et d'affectations des variables, les heuristiques de choix de variables et valeurs y sont indiquées et expliquées. Les paramètres des améliorations du modèle tout comme ceux des algorithmes de résolution exacte et approchée seront précisés et justifiés.

Le dernier chapitre concerne les résultats expérimentaux et l'analyse des résultats obtenus grâce aux différentes améliorations et aux algorithmes de résolution. Nous verrons ainsi l'efficacité des algorithmes implémentés et l'intérêt des diverses améliorations. L'analyse des solutions obtenues rendra compte des spécificités de ce problème.

Chapitre 1

Problématique ferroviaire

Le train est un mode de transport marqué par une forte composante de planification (programmation d'actions et d'opérations à mener). Cette composante s'explique par le coût et la diversité des ressources à utiliser pour la réalisation des services de transport. Par ailleurs, la construction de nouvelles infrastructures, l'engagement de personnel et l'achat de matériel engendrent des coûts colossaux. Ainsi la possibilité d'évaluer la capacité d'une infrastructure (dont une définition précise sera donnée par la suite) permet d'aider le décideur dans le choix des différentes modifications à mettre en œuvre.

Cette évaluation est possible grâce à la résolution du problème dit de saturation de l'infrastructure ferroviaire. Pour une ligne, où l'ordre des trains ne peut pas changer (pas d'arrêt, ni d'aiguille, ni de changement de sens de circulation), le problème de saturation est relativement simple à modéliser et à résoudre. En effet, l'expression théorique de la capacité d'une ligne dans une direction donnée peut être fournie, comme indiqué dans [dCdFU96], par la formule :

$$C = \frac{T}{Ds} \tag{1.1}$$

où T représente la période de l'étude et Ds le temps de succession entre deux trains successifs. Par contre, le cas d'un nœud, d'une gare, impliquant des circulations en sens différents sur une même voie, des jonctions et arrêts possibles, requiert une modélisation plus complexe [Hachemane97].

L'infrastructure ferroviaire étant la principale ressource associée au problème traité, ses caractéristiques sont ici détaillées. Puis l'intérêt de l'évaluation de la capacité ferroviaire est argumenté. Le problème de capacité d'infrastructures ferroviaires est ensuite décrit. Enfin, les principaux travaux sur l'optimisation de l'exploitation de l'infrastructure ferroviaire sont présentés.

1.1 L'infrastructure ferroviaire

Le transport ferroviaire, comme une majorité des transports guidés, requiert trois types de ressources :

1. Le matériel roulant (trains) correspond au convoi ferroviaire circulant sur l'infrastructure. Ce matériel a une durée de vie longue, ce qui implique le besoin d'un parc diversifié aux caractéristiques hétérogènes telles que vitesse, longueur, distance de freinage, ... Il permet le transport de voyageurs et marchandises ;
2. le personnel grâce à ses qualifications (conduite, contrôle des titres de transport, maintenance des voies, ...), permet d'assurer un certain nombre de tâches dont une grande partie contribue à la sécurité de ce mode de transport.
3. l'infrastructure ferroviaire est occupée par le matériel roulant, elle représente l'ensemble des supports physiques de l'activité de transport. L'infrastructure du réseau ferré national comprend la caténaire, les installations de signalisation et les éléments associés (dispositifs d'annonce, de répétition), les voies. Une voie ferrée est un chemin de roulement pour les convois ferroviaires, constitué de plusieurs sections de paires de rails parallèles mis bout à bout.

L'infrastructure ferroviaire peut présenter différentes typologies, pouvant aller d'une simple voie à des ensembles de voies parallèles (tronçons, cf. figure 1.1) ou sécantes (jonction ou nœud, cf. figure 1.3) et des zones d'arrêts (quais). Ces différentes typologies peuvent être classées en deux catégories :

1. une partie homogène où les trains ne peuvent ni changer de voies, ni s'arrêter, ni avoir des sens de circulations différents,
2. une partie hétérogène où des changements de voies, des arrêts, des changements de sens de circulations sont possibles.

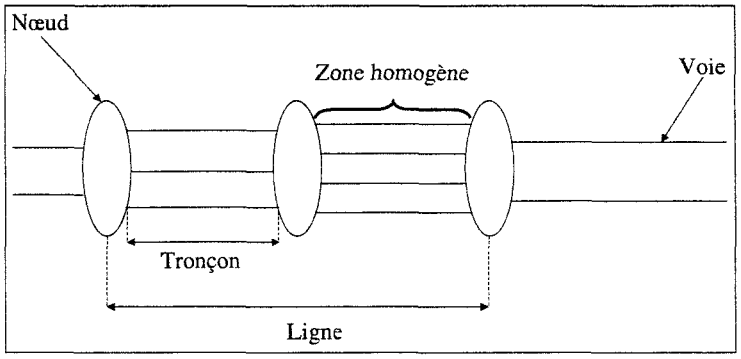

FIG. 1.1 – Exemple d'éléments d'une infrastructure [Delorme03]

Sur ce type d'infrastructure, plusieurs situations de collisions (ou conflits) ou de déraillement des trains existent. La figure 1.1 illustre les principales situations de risques de conflits.

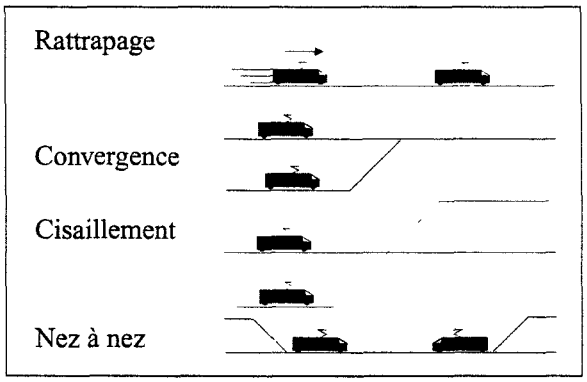

FIG. 1.2 – Typologie des principaux conflits ferroviaires

Les changements de voies, de sens, les arrêts et les différences de vitesses des circulations peuvent être à l'origine de collisions. Pour se protéger contre ces risques, la signalisation ferroviaire s'appuie sur deux principes : le principe d'espacement et le principe d'enclenchement.

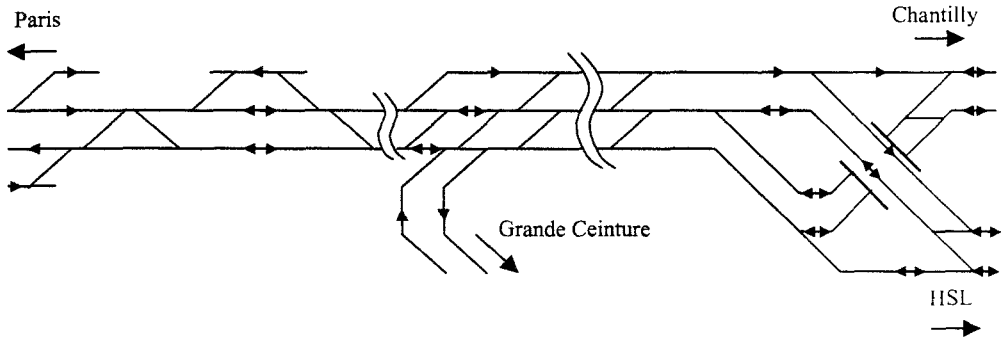

FIG. 1.3 – Nœud de Pierrefitte-Gonnesse

Principe d'espacement

Le principe d'espacement permet de garder, en toutes circonstances, les trains circulant dans un même sens à des distances supérieures à leur distance d'arrêt. Ce principe est obtenu grâce à un système de "cantons". Ce principe se décompose schématiquement en trois étapes :

1. Division de la voie en cantons,
2. Localisation des trains sur l'infrastructure,
3. Envoi de consignes de vitesse à l'entrée de chaque canton qui vérifient en toutes circonstances le principe d'espacement.

La localisation des trains est permise grâce au découpage de la voie en zones de détection. La voie est ensuite décomposée en cantons dont les limites sont matérialisées par des panneaux qui fournissent les consignes de conduite aux conducteurs de convoi (appelé mécaniciens). Il est à noter qu'un canton peut correspondre à une zone, mais il peut également contenir plusieurs zones. Les types de consigne à l'entrée des cantons sont appelés signaux ou aspects de signalisation. Ces signaux indiquent au conducteur sa vitesse maximale autorisée : pleine vitesse, allure réduite, s'arrêter. La figure 1.4 illustre le cas d'un système de signalisation à trois aspects : voie libre : VL (pleine vitesse), avertissement : A (ralentir), sémaphore : S (arrêt).

Le long d'un parcours, même peu étendu, la vitesse des trains et ainsi leur distance peut être contrôlée par un nombre d'aspects différents. Ainsi l'intervalle entre deux circulations peut être réduit en augmentant le nombre d'aspects et en diminuant la longueur des cantons. Mais chaque aspect supplémentaire a un coût de plus en plus important (nombre supérieur de cantons et donc de signalisation et de capteurs sur la voie) pour une augmentation du débit plus faible.

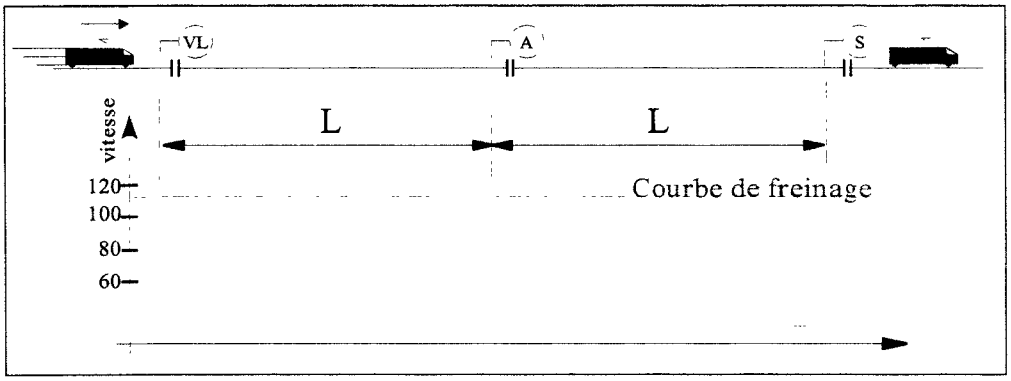

FIG. 1.4 – Principe d'espacement

Principe d'enclenchement

Le principe d'enclenchement vise à imposer des conditions aux manoeuvres des appareils de voie (e.g. un aiguillage) et aux signaux de protection. Le nombre et la complexité des situations concernées ne peuvent être repris ici, nous nous limiterons à un cas illustré par la figure 1.5. Dans cet exemple, on s'intéresse au passage des circulations sur une voie banalisée (i.e. avec deux sens de circulation possibles). Le train 1 doit suivre l'itinéraire p_1 et le train 2 doit suivre l'itinéraire p_2. Si le train 1 doit passer avant le train 2, il faut alors commencer par former l'itinéraire du train 1, c'est-à-dire positionner les deux aiguillages pour former l'itinéraire p_1 ainsi que les signaux en voie libre pour cet itinéraire. Un signal protège l'aiguillage de la zone z_a (une zone est délimitée par deux petits traits sur la figure) et empêche le train 2 d'accéder à la partie de voie banalisée (zones communes entre z_a et z_b). On dit que la position de l'aiguille de la zone z_a enclenche ce signal, c'est-à-dire qu'il est impossible de mettre ce signal en voie libre au moins tant que l'aiguille

reste dans la même position. Une fois le train passé, le signal pourra passer en voie libre lorsque le train 1 aura libéré la zone z_b (le train 1 aura donc quitté la zone commune).

Lors de l'élaboration des grilles horaires, ce principe est pris en compte afin de calculer la marge temporelle qu'il faut prévoir entre les passages de deux trains qui circulent sur des itinéraires qui ont une zone commune. En phase opérationnelle, ce sont les signaux sur la voie qui permettent d'assurer l'application de ces principes.

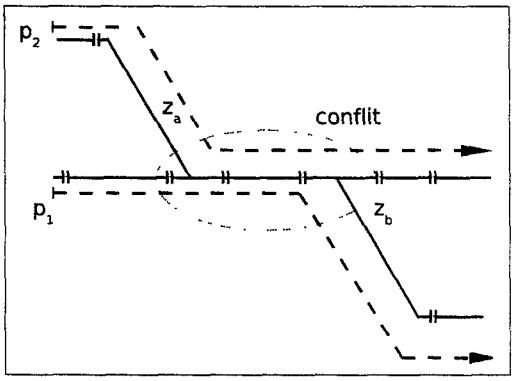

FIG. 1.5 – Un conflit de contre sens

1.2 Intérêt de l'évaluation de la capacité

Le transport ferroviaire, comme d'autres domaines de gestion de la production (se reporter à [Esquirol et al.01] pour une étude complète de la gestion de production), possède une forte composante de planification. La planification ferroviaire se décompose en trois niveaux de décision en fonction de leur échéance temporelle de mise en œuvre :

1. *Le niveau stratégique* se place dans le long terme (jusqu'à une cinquantaine d'années). Plusieurs projets d'aménagements amènent à prendre des décisions importantes, telles que la construction ou la modification d'infrastructure, l'abandon d'une partie des installations, le recrutement de personnel et l'achat de matériel roulant. Ces aménagements doivent être évalués et décidés bien avant leur mise en œuvre. C'est donc dans la définition de la stratégie de l'entreprise que se situe ce type de décision.
2. *Le niveau tactique* est situé dans le moyen terme. Il s'agit de définir un plan de transport permettant de répartir les trains dans l'infrastructure afin de suivre au mieux la demande. À ce niveau, l'utilisation des ressources (personnel, infrastructures, trains, ...) est répartie dans le temps et l'espace.
3. *Le niveau opérationnel* se place dans le court terme (journée). Il correspond à la mise en œuvre et à la modification du plan de transport défini au niveau tactique. En effet, les ho-

raires théoriques ne sont pas complètement applicables en phase opérationnelle, à cause notamment des aléas de disponibilités des ressources (locomotive tombant en panne la veille ou incidant de circulation). Ce type d'aléas implique alors des adaptations des horaires établis.

Le tableau 1.1 résume les trois niveaux de décisions.

Il est à noter que les problèmes liés à la gestion prévisionelle des aléas, tels que les retards, n'entrent pas dans le cadre de ces niveaux, mais dans celui du pilotage des ressources en temps réel.

Domaine	Planification	Programmation	Ordonnancement
Niveaux de décisions	Stratégique	Tactique	Opérationnel
Horizon temporel	années	mois	journées
Gestion	– Infrastructures – Personnel – Offre de transport	– Horaires – Roulements du personnel – Roulements du matériel	– Gestion des dépôts – Gestion des circulations – Maintenance

TAB. 1.1 – Niveaux de décisions en gestion prévisionnelle des modes de transport en commun

Le niveau stratégique est le niveau essentiel pour l'avenir du réseau ferré. Toute modification d'infrastructure implique un coût très lourd et engendre un impact irréversible sur les ressources disponibles et donc sur l'offre. Il est donc primordial de se doter d'outils permettant d'évaluer les limites du réseau existant et de simuler l'impact des modifications proposées.

En ce qui concerne l'évolution du trafic, deux cas peuvent se présenter :

- Si la demande est en hausse, il faut être en mesure de savoir si les installations existantes peuvent supporter une augmentation du trafic, si nécessaire quelles parties du réseau améliorer et proposer des solutions d'amélioration.
- À l'inverse le cas d'une diminution de la demande implique une restructuration du réseau, il est alors intéressant de voir quelles parties des installations ne plus utiliser.

Dans le contexte actuel, la demande pour le transport ferroviaire (de passagers et de marchandises) est de plus en plus forte, et ce pour deux raisons principales :

- Le mode ferroviaire se positionne comme un moyen de substitution à la voiture. En effet, son coût est relativement faible. De plus, il est un très bon moyen de désengorger les routes et de diminuer la pollution automobile.
- La mobilité humaine et matérielle (transport de marchandises) augmente de plus en plus. Les modes de transports doivent alors s'adapter à ces nouvelles demandes afin d'absorber un trafic plus important et diversifié (TGV, trains grandes lignes, trains de marchandises, ...).

À cause de cette accentuation du trafic, certains nœuds et tronçons du réseau ferroviaire sont déjà très saturés et se transforment en véritables goulets d'étranglement. Les retards s'en trouvent alors amplifiés et la gestion du trafic devient critique, voire impossible. L'évaluation de la capacité

d'une infrastructure est un moyen de mettre en évidence les limites du trafic, elle permet donc de pouvoir traiter le problème plus tôt.

La directive européenne [Cce] implique la séparation de la gestion de l'infrastructure et de son exploitation. Ainsi, en France, c'est au gestionnaire de l'infrastructure, RFF (Réseau Ferré Français), de pouvoir fournir aux exploitants, la SNCF, des ressources suffisantes pour répondre à la demande.

L'évaluation de la capacité ferroviaire s'avère donc d'une importance cruciale pour l'avenir du transport ferroviaire afin de répondre à la demande croissante.

1.3 Évaluation de la capacité ferroviaire

1.3.1 Définition de la capacité ferroviaire

La capacité correspond au trafic maximal pouvant circuler sur un élément. Pour un flux, la capacité peut être vue comme le débit maximal sous une série de contraintes. Dans le cadre ferroviaire, le flux n'est pas continu mais discret, dépendant des trains. La capacité d'une infrastructure ferroviaire correspond alors au maximum de trains pouvant circuler sur cette infrastructure en un temps donné. Il n'existe pas de définition unique de la capacité d'une infrastructure ferroviaire, celle-ci dépendant fortement des hypothèses du problème. La validité d'une étude de capacité dépend donc du cadre de ces hypothèses. Nous avons retenu 2 définitions de la capacité d'infrastructure ferroviaire :

Définition selon Bellaiche

Elle distingue une capacité théorique représentant la capacité maximale atteignable techniquement et une capacité pratique prenant en compte les aléas de l'exploitation.

Définition 1 (Capacité d'une infrastructure ferroviaire [Bellaiche97]). *La capacité théorique est le nombre maximum de trains que l'on peut faire circuler théoriquement sur une ligne ou à travers un nœud pendant une période donnée. La notion de capacité pratique quant à elle, tient compte d'une marge dite de souplesse afin d'éviter la saturation totale et les retards en cascade en cas d'incident.*

Cette définition montre la nécessité de délimiter le périmètre de l'étude sur le plan spatial et temporel. Elle révèle une capacité uniquement par rapport à l'infrastructure. Beaucoup d'autres paramètres influent sur la capacité, comme décrit dans [Schneider97] et [Bellaiche97] :

- le type d'infrastructure tel que le nombre de voies, la vitesse autorisée, les aiguillages,
- le plan de transport, c'est à dire l'ordonnancement des trains, les contraintes horaires,
- les caractéristiques techniques des trains (matériel hétérogène).

Le point de vue des clients du service de transport peut aussi être considéré en prenant en compte la qualité de service (notion difficile à définir et quantifier car différente selon les clients).

Définition selon Hachemane

Définition 2 (Capacité d'une infrastructure ferroviaire [Hachemane97]). *La notion de capacité peut se définir comme le nombre maximum de trains pouvant circuler dans un intervalle de temps donné dans des conditions pratiques d'exploitation et pour une structure de lignes, une structure d'horaire et une qualité de service données.*

Cette définition implique différents facteurs à définir :

- Nombre de trains : cette valeur correspond à la quantité de trains empruntant, sans interruption, tout ou une partie de l'élément d'infrastructure considéré.
- Structure de lignes : Elle définit et délimite le réseau considéré pour l'étude de la capacité. Cela correspond à une notion d'infrastructure et d'exploitation.
- Structure de l'horaire : Il s'agit de connaître la répartition à l'horaire des différents types de trains et les liens entre les trains (i. e. les cadences, les correspondances, les trains en batteries, ...).
- Conditions d'exploitation : Elles sont supposées être pratiques, c'est à dire qu'elles impliquent une majoration des temps d'occupation par rapport au temps théorique afin de tenir compte des perturbations qui restent inéluctables dans l'exploitation.
- Qualité de service : C'est une notion très difficile à définir. Elle était traditionnellement présentée sans le client, car elle visait principalement l'optimisation de la rentabilité [Felici et al.92]. L'identification correcte nécessite des actions de contrôle continues visant à conforter les décisions stratégiques et les choix de projets par une connaissance précise des usagers (aussi bien actuels que potentiels), mais aussi à une prévision fiable de l'évolution de la clientèle à moyen et long termes. À titre d'exemple pour la clientèle, la qualité de service pourrait correspondre à son degré de satisfaction (confort, horaires). Les composantes de qualité de service influant directement sur la capacité sont essentiellement les temps de parcours, les fréquences (nombre de trains sur une liaison en un temps donné) et la ponctualité (fonction du retard à l'arrivée du train).

Il est à noter que cette définition de capacité est plus complète que la précédente car elle tient compte des conditions pratiques et non théoriques, de la structure de l'infrastructure, du matériel et de la qualité de service. Nous utiliserons donc la définition proposée par Hachemane dans cette étude.

1.3.2 Le problème de saturation

La capacité d'un réseau ferré complet ou même partiel ne peut pas être déterminée par une seule mesure. En effet, la capacité d'une zone homogène tel un tronçon peut être obtenue relativement

simplement en effectuant la somme des capacités des différentes voies du tronçon. Mais la capacité d'une zone hétérogène ne permet pas l'additivité des capacités de chaque ligne (à cause notamment de croisement et zones de voie empruntable dans les 2 sens de circulation). De ce fait, il convient de séparer l'infrastructure en plusieurs parties, ayant chacune une mesure de capacité différente, comme préconisé dans [Pachl04] :

- tronçon,
- nœud, c'est à dire un ensemble de lignes, aiguilles, gares, ... représentant un point critique du réseau.

La capacité d'une zone hétérogène peut être obtenue par la solution d'un problème d'optimisation appelé problème de saturation et défini comme suit :

Définition 3 (Problème de saturation [Delorme03]). *Le problème de saturation consiste à introduire le maximum de circulations supplémentaires dans une grille horaire établie (éventuellement vide). Les trains ainsi ajoutés représentent la marge de capacité disponible (c'est à dire la capacité résiduelle, ou la capacité absolue lorsque la grille horaire établie est vide) de l'infrastructure par rapport à une offre.*

1.3.3 Problèmes connexes à l'évaluation de la capacité

Une étude de capacité ne se résume pas simplement à établir une mesure, elle amène plusieurs questions et, comme indiqué dans [Delorme03], notamment :

- La capacité de l'infrastructure est-elle suffisante pour faire face à l'évolution de l'offre ?
- Quels sont les meilleurs investissements du point de vue capacitaire ?
- Quelle est la meilleure façon de répondre à l'offre proposée par le décideur (offre directement liée à la demande des clients) ?

Le problème de faisabilité

La première question correspond à un autre problème appelé problème de faisabilité. Il s'agit ici de savoir si l'offre future est réalisable. La définition de ce problème est la suivante :

Définition 4 (Problème de faisabilité [Delorme03]). *Le problème de faisabilité consiste à trouver les itinéraires d'un ensemble de trains pour que le parcours sur l'infrastructure étudiée se réalise sans retard et avec un niveau de robustesse (marge permettant de minimiser l'impact de perturbations) donné.*

Le problème de saturation peut être vu comme le prolongement du problème de faisabilité en introduisant le maximum de circulations supplémentaires sur la grille horaire donnée. Les trains ajoutés représentent alors la marge de capacité disponible de l'infrastructure par rapport à une offre établie.

Choix d'investissement

Le but de la seconde question est de déterminer les meilleurs investissements permettant d'accroître la capacité de l'infrastructure. Il s'agit d'étudier l'impact de modifications d'un ou plusieurs élément(s) de l'infrastructure (variantes d'infrastructures, de solutions). Cette question reprend les problématiques de faisabilité et de saturation. Auquelles s'ajoutent la nécessité de déterminer les points bloquants de l'infrastructure, c'est-à-dire ceux qui limitent la capacité. C'est en effet à ce niveau que les investissements devront être effectués.

Le problème des préférences

Enfin, la dernière question correspond à la notion difficile à définir de la qualité de service. Elle couvre ainsi plusieurs problèmes d'optimisation du passage des trains dans l'infrastructure. Elle peut être envisagée tant du point de vue du client que du décideur. Le décideur exprime ses préférences en fonction des attentes des usagers (supposées ou exprimées) :

Définition 5 (Problème d'optimisation des préférences [Delorme03]). *Le problème d'optimisation des préférences consiste à déterminer un routage et ordonnancement maximisant la somme des préférences sur des choix de parcours et/ou de dates lors du passage d'une combinaison donnée de trains (point et date d'entrée-sortie fixés) dans une infrastructure considérée.*

Dans ce cadre se situe également le problème de la fluidification du passage de ces trains lorsqu'il n'est pas possible de satisfaire l'offre proposée :

Définition 6 (Problème de fluidification [Delorme03]). *Le problème de fluidification consiste à déterminer un routage et ordonnancement minimisant la somme des retards générés lors du passage d'une combinaison donnée de trains (point et date d'entrée-sortie fixés) dans une infrastructure considérée.*

Cependant, ce dernier problème relève plus d'un niveau de décision opérationnelle que de planification stratégique ou tactique, il n'intéresse donc pas notre étude.

La description du problème de saturation d'une infrastructure ferroviaire a été donnée, la partie suivante présente les principales méthodes existantes pour évaluer la capacité d'une infrastructure ferroviaire.

1.4 Évaluation de la capacité : travaux existants

Les problèmes soulevés dans le domaine ferroviaire ainsi que les modèles d'optimisation proposés sont nombreux. Les principaux problèmes de routage et d'ordonnancement sont repris dans [Bussieck et al.97] et [Cordeau et al.98]. L'évaluation de la capacité d'une infrastructure ferroviaire peut être effectuée à plusieurs niveaux :

- niveau microscopique, telle que la capacité d'une ligne, d'un nœud ou d'une gare,
- niveau macroscopique, tel que le réseau ou une partie du réseau comprenant plusieurs nœuds (ligne Paris-Lille par exemple).

Les méthodes indiquées ci-après, représentent l'ensemble des études envisagées pour l'évaluation de la capacité.

Les méthodes analytiques Elles permettent d'obtenir une valeur théorique représentant la capacité de l'infrastructure étudiée. Elles ont principalement pour origine la formule issue d'une fiche éditée par l'UIC (Union Internationale des Chemins de fer) [dCdFU96]. L'idée consiste à utiliser le temps minimal de succession entre trains. Elle se base sur la capacité de la section de ligne déterminante, c'est à dire celle ayant la capacité la plus faible. La section de ligne déterminante est celle où les temps de parcours sont les plus élevés et la durée moyenne de succession des trains est constatée la plus grande. Ces sections agissent telles un goulet d'étranglement. La première étape consiste à découper le tronçon en sections de lignes homogènes. Puis la section de ligne déterminante est identifiée. Ensuite la formule de calcul de capacité suivante est appliquée :

$$L = \frac{T}{t_{fm} + t_r + t_{zu}} \tag{1.2}$$

avec T : période de référence de l'étude (1 heure), t_{fm} durée moyenne de succession des trains, t_r marge de détente, t_{zu} temps supplémentaire. Ces deux derniers temps sont obtenus par l'expérience sur le terrain. La marge de détente correspond à :"un temps supplémentaire prévu après chaque durée de succession minimale pour réduire le risque d'apparition des retards en cascade".

L'avantage de cette méthode UIC comme celles qui s'en sont inspirées sont leurs simplicité et rapidité de mise en œuvre. En contrepartie leurs hypothèses sont souvent trop réductrices.

Les méthodes probabilistes Elles se basent sur la probabilité d'utilisation des ressources [Florio et al.96]. Elles se basent sur la probabilité que deux trains empruntent la même ressource, cette probabilité va influer directement sur le coefficient d'utilisation de la ressource. La capacité des différentes ressources est ensuite obtenue par une méthode analytique. Puis il s'agit de maximiser le nombre de trains de chaque type de trafic (ensemble de trains ayant un sens, origine, destination différente) sur la ressource. Cette maximisation se fait sous les contraintes :

- qu'un minimum et maximum de trains pour chaque courant soient respectés,
- que la somme des temps consommés soit comprise dans l'intervalle de temps considéré.

Ce genre d'approche permet d'obtenir des solutions acceptables pour un élément isolé si le trafic est réparti régulièrement sur l'intervalle horaire. La valeur obtenue est théorique et impose des hypothèses souvent réductrices, mais elle a l'avantage d'être relativement simple à mettre en œuvre.

Les méthodes de simulation Elles sont parfois utilisées pour "évaluer" le degré d'utilisation de la capacité, on peut notamment citer RWS[1] . Elles ne proposent pas de calcul à partir de paramètres globaux de l'infrastructure et de grilles horaires mais elles simulent pour des infrastructures données, la mise en œuvre de grilles horaires des conditions proches de la réalité (i.e. qui incluent l'impact d'aléas de l'exploitation). Le principe sur lequel reposent ces méthodes est que la capacité d'une infrastructure est atteinte lorsque l'application d'une grille horaire devient instable. Ces méthodes donnent donc une définition de la capacité qui est très liée aux grilles horaires utilisées et à la façon de les mettre en œuvre dans les simulations.

Les méthodes constructives Elles élaborent, depuis une grille horaire, une grille la plus dense possible. Elles peuvent être efficaces (résolution rapide et solution de bonnes qualité) même sur des zones assez complexes. Si les données sont suffisantes, l'horaire obtenu sera proche d'un horaire applicable. Cette applicabilité dépend de la finesse de la modélisation : plus elle est fine, plus le temps de résolution est grand. La question de la prise en compte de la qualité de service peut également se poser du fait de la saturation du réseau et donc du manque de stabilité de l'horaire. Dans ce cas l'utilisation d'un simulateur peut être préconisée. Le principal inconvénient de ce genre de méthode est donc soit d'ommettre des conflits (si la modélisation n'est pas assez fine, certains conflits ne pourront pas être vus) dans des nœuds, soit de nécessiter un volume d'information très important. En effet, plus la modélisation est fine et plus il faudra de données pour représenter chaque possibilité de modélisation.

Parmi l'ensemble de ces méthodes, seules les méthodes constructives permettent d'assurer la faisabilité d'une grille horaire saturée, c'est pourquoi il a été décidé de choisir une méthode constructive pour notre problème. Par la suite, les principales méthodes appartenant à cette dernière catégorie seront détaillées.

1.4.1 DONS (Design Of Network Schedules)

Le projet DONS a été développé par les chemins de fer néerlandais[2]. Une première présentation en a été faite par [Van den berg et al.94], puis [Zwaneveld97] en a donné une description plus précise. Il a pour objectif d'apporter des aides pour la planification d'une ou plusieurs grille(s) horaire(s) cadencée(s) sur l'ensemble du réseau hollandais afin d'évaluer la capacité de projets d'aménagement. L'aboutissement de ce projet résulte en deux modules complémentaires, comme le présente la figure 1.6. Le premier, CADANS, développé par Schrijver et Steenbeek [Schrijver et al.94], aide le planificateur à établir une grille horaire cadencée (la même grille horaire peut se répéter toutes les heures) à une heure sur l'ensemble du réseau. Le second, STATIONS

[1]Railway Simulation développé à l'origine par [Giger87]
[2]Nederlandse Spoorwegen (NS)

(développé par Zwaneveld et al. [Zwaneveld et al.96]), permet de résoudre les problèmes de routage à l'intérieur des gares afin de vérifier la faisabilité de la grille horaire proposée par CADANS. Si STATIONS ne trouve pas de solution, alors il met en évidence les trains bloquants pour CADANS qui régénèrera une grille horaire ou augmentera les marges admissibles sur les horaires des trains. Lorsque CADANS ne parvient pas à trouver de grille horaire réalisable, l'infrastructure doit être modifiée ou l'offre doit être diminuée. L'information sur les points du réseau et la densité du trafic posant problème est donnée à l'utilisateur du système : c'est alors à lui de modifier l'infrastructure ou/et de réduire l'offre.

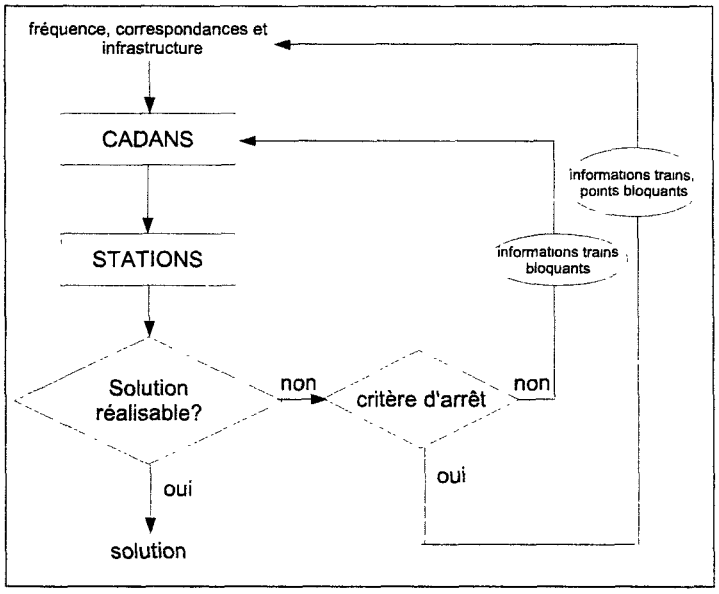

FIG. 1.6 – Schéma du projet DONS

CADANS [Schrijver et al.94]

Son objectif est d'élaborer une grille horaire cadencée à une heure pour l'ensemble du réseau. La solution ne tient pas compte des contraintes de capacité sur les gares, ces dernières ont en quelque sorte une capacité infinie. Ainsi ce module ne tient compte que des lignes situées entre les gares.

La grille horaire établie est caractérisée par les heures d'arrivées et de départs des trains dans chaque gare. Les contraintes auxquelles sont soumises les variables de décisions portent :

- sur les temps de parcours entre les gares,
- le temps de présence de trains en gare, devant permettre la correspondance s'il y a lieu,
- l'utilisation de l'infrastructure, deux trains utilisant la même partie de l'infrastructure doivent avoir un écart de temps suffisant (gestion des conflits).

Les méthodes de résolutions font appel aux techniques de programmation par contraintes, Branch & Bound et à des méthodes de coupes. La résolution est alors effectuée dans un temps raisonnable (plusieurs tests d'infrastructures pouvant être effectués en quelques semaines au moment de l'étude) pour la majorité des cas, mais il existe des cas où la résolution exacte n'est pas possible.

STATIONS [Zwaneveld et al.96]

Ce module permet de résoudre les problèmes de routage dans chaque nœud afin de vérifier la faisabilité de la grille horaire proposée par CADANS. Si ce module émet un échec sur le plan de transport proposé par CADANS, il met en évidence les trains bloquants. Et ainsi CADANS va réaliser une nouvelle grille horaire en considérant ces trains bloquants. Le modèle mathématique proposé pour résoudre ce problème est le problème de Set Packing [3] transformé en Node Packing [4] pendant la phase d'initialisation. La résolution du problème contient trois phases :

1. La phase d'initialisation détermine les itinéraires compatibles et donc les variables indépendantes. Les sections considérées sont celles contenant un aiguillage, une intersection, un point d'entrée, un quai ou un point de sortie. Cela suppose que l'infrastructure ne contient pas de voies banalisées et qu'aucun croisement n'a lieu. De plus, le temps de parcours des sections est supposé identique, ce qui exclut les dépassements. Cette hypothèse peut être gênante dans le cas d'un trafic hétérogène, surtout si les vitesses des trains sont très différentes.
2. La phase de réduction permet d'éliminer des contraintes et des variables inutiles grâce à une phase de test de pertinence des variables et à quatre tests valides pour les problèmes de node packing (réduction des contraintes et variables de l'ordre de 93%).
3. La phase de résolution utilise un algorithme de Branch & Cut.

Le système DONS est utilisé pour générer plusieurs grilles horaires comportant des structures différentes. Le nombre de solutions de qualités différentes indique la flexibilité du réseau considéré. Plusieurs variantes de l'infrastructure peuvent également être testées, dans ce cas, les solutions sont proposées après un temps de calcul ne dépassant pas quelques semaines. La robustesse des solutions proposées peut être évaluée en utilisant un modèle de simulation de réseau.

1.4.2 CAPRES (système d'aide à l'analyse de la CAPacité des RÉSeaux ferroviaires)

Les chemins de fer Suisses[5] ont développé un outil fournissant un système d'aide à l'évaluation de la capacité de réseaux complets par construction d'horaires. Ce projet s'inscrit dans les travaux

[3] problème d'optimisation combinatoire à variables booléennes ne comportant que des contraintes de disjonction
[4] set packing dont les contraintes ne considèrent que des couples de variables
[5] Chemins de Fer Fédéraux suisses (CFF)

d'Hachemane [Hachemane97]. Ce système permet de traiter deux types de problèmes :

P_h : un problème d'élaboration de variantes d'horaires cadencés qui doit permettre de faire circuler les trains en toute sécurité sur l'infrastructure considérée. Ces variantes d'horaires sont dénommées variantes de base,

P_s : un problème d'évaluation de la capacité de l'infrastructure en saturant une variante de base choisie. Cette saturation se fait grâce à une liste de trains saturants pouvant avoir des priorités préalablement définies.

Le réseau est modélisé en un graphe (nœuds et arcs). Les nœuds correspondent aux points singuliers du réseau (gare, jonction, ...). Les arcs, orientés, correspondent aux voies de tronçon du réseau. L'hypothèse fondamentale du découpage est que l'ordre des trains ne peut pas être modifié sur un arc, cela implique que tous les points de bifurcation, gares de croisement, de terminus et de dépassement doivent être modélisés par des nœuds. Les trains sont décrits par leur parcour et horaire. Le parcours est défini par un ensemble de nœuds reliés entre eux par des arcs orientés, le nœud origine correspond au point de départ, le nœud final à la destination des arcs. L'horaire du train, quant à lui, est défini uniquement au niveau des nœuds du parcours du train. Les trains ayant le même parcours sont regroupés en famille, les trains d'une même famille diffèrent par leur horaire.

Les nœuds sont modélisés par trois niveaux de détails différents :

1. aucun conflit dans le nœud n'est pris en compte,
2. seuls les conflits de cisaillement sont pris en compte,
3. tous les conflits sont pris en compte (y compris ceux d'attribution de quais).

Ces données permettent de laisser aux trains saturants une certaine souplesse (modélisation de la marche détendue, possibilité d'arrêts facultatifs pour un dépassement, par exemple).

Dans le problème traité, l'objectif n'est pas d'obtenir un horaire mais plutôt d'obtenir les bornes maximales sur les dates d'événements d'un horaire compte tenues d'un certain nombre de contraintes temporelles. L'auteur distingue deux types de contraintes temporelles :

- Les contraintes dites potentielles indiquent qu'un événement F (débutant à h_F) doit se produire dans un intervalle de temps dépendant de l'événement E (débutant à h_E) et avec une marge (t^-, t^+) :

$$h_E + t^- \leq h_F \leq h_E + t^+, \tag{1.3}$$

  avec $t^- \geq 0, t^+ \geq 0$.
- Les contraintes disjontives précisent les conditions qui séparent les dates de deux événements E et F suivant les deux séquencements possibles (i.e. E avant F ou F avant E avec un écart de t_1 ou t_2) :

$$h_E + t_1 \leq h_F \vee h_F + t_2 \leq h_E \tag{1.4}$$

Pour la résolution du problème d'élaboration P_h, la méthode s'inspire de l'algorithme de Fukumori et Sano [Fukumori et al.87]. Cet algorithme ne s'applique qu'au cas d'une ligne à deux voies et ne considère le problème que voie par voie et non train par train. La méthode utilisée par l'algorithme de CAPRES est décomposée en deux phases : tout d'abord les disjonctions sont résolues, puis vient la propagation des contraintes. L'algorithme utilisé consiste à placer, un par un et voie par voie, chacun des trains de l'horaire (à chaque fois qu'un ordre de succession a été fixé entre deux trains sur une voie de tronçon). Une fois l'ordre de succession fixé, la contrainte disjonctive résolue est propagée. L'algorithme procède train par train et voie par voie, il est donc primordial de commencer la recherche par les trains les plus contraints.

En ce qui concerne le problème de saturation P_s, le même algorithme que pour le problème P_h est utilisé, mais cette fois de manière itérative en introduisant à chaque pas un ou plusieurs train(s) de la liste saturante considérée. Les trains ajoutés dépendent de la stratégie de saturation choisie.

Le système CAPRES est constitué principalement de deux logiciels :

- INSCAP : permet l'introduction et la modification d'une base de données, la production de résultats par vérification, élaboration, ajout, suppression de trains ou calcul de symétrie.
- RESCAP : permet l'édition sous différentes présentations classiques d'un horaire ferroviaire produit par INSCAP.

INSCAP

Les données (fichiers) d'entrées pour ce logiciel sont :

- les données du réseau,
- une liste d'élaboration de variantes d'horaires (trains, horaires, parcours),
- une liste saturante (ensemble de trains pouvant être introduits pour la saturation du réseau) et sa stratégie d'introduction,
- une variante d'horaire (succession différente des trains) de base ou saturé.

Une fois les données entrées, INSCAP permet :

- La modification des données du réseau de CAPRES sur 8 ensembles : nœuds, tronçons, familles (série de trains ayant des paramètres communs), sillons, espacements entre deux trains successifs, correspondances, fréquences et itinéraires.
- L'élaboration d'horaire, à partir de la liste d'élaboration des variantes d'horaire (trains, itinéraires, horaire).
- La saturation d'horaire. Cela concerne la saturation d'une variante d'horaire de base avec une liste saturante et selon une stratégie spécifiée.
- La vérification d'horaire : Lorsqu'aucune variante d'horaire n'a été obtenue, il est utile de savoir pourquoi. Ce module relâche les contraintes disjonctives, ce qui permet d'indiquer, pour chaque contrainte relâchée, la valeur nécessaire à introduire pour que l'horaire soit respecté ainsi que la valeur permettant l'élaboration de variantes d'horaire.
- L'ajout ou suppression de trains à l'horaire : L'horaire peut être modifié si la solution pro-

posée par CAPRES ne répond pas aux souhaits de l'utilisateur. La modification s'opère par l'ajout ou la suppression de trains. L'ajout d'un train correspond approximativement à la saturation qui stopperait dès l'ajout du premier train. La suppression de trains revient à éliminer de la variante d'horaire une série de trains spécifiés.

- La symétrie d'un horaire cadencé : il s'agit de variantes dans lesquelles des couples de trains ayant un parcours opposé (l'origine de l'un est la destination de l'autre) sont formés tel que ces trains se croisent en un nœud au milieu de la cadence. Ceci permet donc d'avoir des nœuds de correspondances.

RESCAP

Le logiciel RESCAP est un éditeur graphique et alphanumérique d'horaire. Les données nécessaires sont donc le réseau et l'horaire. Ce logiciel est indépendant de INSCAP et peut permettre l'édition d'horaires produits par d'autres logiciels ou manuellement.

Outre les présentations d'horaires, plusieurs fonctionnalités sont directement en rapport avec la capacité :

- l'affichage du nombre de trains pour chaque élément du réseau,
- l'affichage des points critiques du réseau (goulets).

Les performances de cette méthode dépendent fortement des contraintes introduites et de la taille du réseau. Pour le problème de capacité, il faut une heure de calcul (processeur à 66MHz) pour une infrastructure à 200 nœuds (cf. [Hachemane97]), le détail des finesses de modélisations n'est pas communiqué. Les facteurs influant sur la résolution sont :

- la définition et la saturation de l'horaire,
- le détail des nœuds.

Le logiciel résultant, décrit par [Curchod et al.01], est également utilisé par d'autres pays et notamment en France pour certaines études.

1.4.3 DÉMIURGE

DÉMIURGE se veut être un outil d'optimisation et de mesure de la capacité du réseau ferroviaire. Il est développé par la SNCF et a été succintement détaillé par Labouisse et Djellab dans [Labouisse et al.01] et [Labouisse et al.02]. Le logiciel est utilisé par l'Ingénierie SNCF et a pour objectif de mettre au point un outil d'aide à la réalisation des études de capacité du réseau ferroviaire. Le but de ce projet est de remplacer le système CAPRES. DÉMIURGE permet notamment :

- d'évaluer un réseau sur l'absorption de nouveaux trafics,
- d'identifier les goulets d'étranglement,
- de trouver les meilleurs investissements sur l'infrastructure,
- d'optimiser les grilles horaires,

– de calculer la capacité résiduelle d'une grille horaire.

Le réseau étudié est décrit par l'utilisateur sous forme de nœuds et de tronçons. Les niveaux de modélisation des nœuds sont les mêmes que ceux proposés dans CAPRES (cf. section 1.4.2). Les trains sont répartis en trois sous-ensembles : les trains de base, les trains demandés pour le court ou le moyen terme et les trains de saturation (demande à long terme). La description des trains porte sur les contraintes commerciales (dates d'entrée-sortie, temps d'arrêt, correspondances,...). La modélisation proposée est un problème en nombre entier (MIP : Mixed Integer Programming) multicritère traité en monocritère. Il est ainsi résolu de manière exacte ou approchée en considérant l'un des critères suivants :

– maximiser le nombre de trains,
– minimiser le temps d'occupation de l'infrastructure,
– minimiser le décalage de l'horaire par rapport à l'horaire de base,
– minimiser le nombre de contraintes d'exploitations violées.

Pour la phase de résolution, des pré-traitements et l'ajout de coupes sont effectués. La résolution du problème est effectué grace au logiciel d'optimisation commercial Cplex d'ILOG. Une méthode de résolution par décomposition est envisagée pour améliorer la résolution. Les résultats obtenus le sont dans des temps satisfaisants pour les tailles de problèmes testés. Par exemple, une instance sur la ligne C du RER de Paris, avec 45 nœuds, 59 tronçons, 24 trains et une plage horaire de 2 heures 30, a été modélisée par 2 303 variables et 6 731 contraintes et une solution a été obtenue en 2 minutes (type de machine non communiqué).

1.4.4 RECIFE

Le projet RECIFE permet l'évaluation de la capacité ferroviaire au niveau d'un nœud. Il se distingue des autres études de capacité par la finesse de modélisation. Ce choix se justifie par la taille des infrastructures considérées. En effet, dans le cas d'un nœud ou d'une gare, le trafic est très dense et l'espace réduit. Dans ce cas, des approximations sur la durée de conflit risquent d'aboutir à des évaluations erronées, en générant de fausses incompatibilités. Ainsi ces projets se placent en complément d'outils d'analyse de réseau tels que CAPRES ou DÉMIURGE qui considèrent un niveau de modélisation plus large (au niveau d'un réseau en entier).

Dans le cadre du projet RECIFE, la contribution de la thèse de Xavier Delorme [Delorme03] porte sur une modélisation linéaire mono et multiobjectif de ce problème sur la proposition et l'expérimentation d'une méthode de résolution exacte et approchée. Cette thèse propose deux structures correspondant à des problèmes d'optimisation classiques appelés plus court chemin [6] et set packing.

Une interface graphique (voir [Marliere01] et figure 1.7 pour un exemple d'interface) a été développée afin d'aider l'utilisateur sur l'étude de capacité. Elle permet également de mettre en

[6] basé sur la notion de graphe et dont l'objectif est de trouver un chemin de valeur totale minimum d'un sommet du graphe à un autre

œuvre une méthode interactive d'aide à la décision en prenant en compte différents critères.

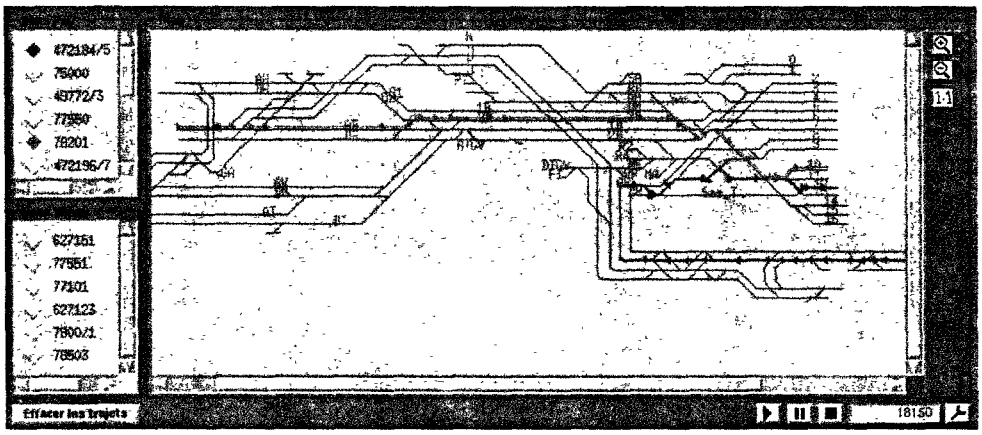

FIG. 1.7 – Schéma détaillé d'une voie ferrée du projet RECIFE

Comme indiqué précédemment, ce projet, portant sur l'analyse de la capacité ferroviaire au niveau d'un nœud, se distingue par une finesse plus importante de la discrétisation du temps par rapport aux autres méthodes. Ainsi les approximations tolérées dans les autres méthodes ne le sont pas ici.

Le problème de saturation modélisé s'inspire des travaux du projet DONS [Zwaneveld97] vu en section 1.4.1. Il est modélisé sous la forme d'un programme mathématique de type set packing (cf. [Delorme et al.04]).

Plusieurs algorithmes de pré-traitements et de résolution approchée sont proposés pour ce problème. Les pré-traitements considérés sont les suivants :

- trouver un ensemble indépendant de variables propres au problème de capacité : cliques,
- calcul de cliques maximales,
- test de dominance entre variables.

Les algorithmes de résolutions proposés sont les suivants :

- La résolution exacte utilise le logiciel commercial Cplex [Ilog99].
- La résolution approchée utilise la métaheuristique GRASP [Delorme et al.04], cette métaheuristique sera décrite plus en détail dans la section 4.4.2.

Les aspects multiobjectifs[7] de ce problème a été réalisé lors du DEA de Degoutin [Degoutin02] puis amélioré et étendu grâce à la collaboration avec Delorme et Gandibleux. Le but de la résolution multiobjectif est de pouvoir optimiser non seulement le nombre de trains mais également d'autres critères simultanément, tels que :

- la préférence des trains, il s'agit de maximiser la somme pondérée des variables du problème en fonction des préférences sur les catégories de trains. Par exemple, faire passer plus de

[7]Pour une bibliographie complète sur les problèmes multiobjectifs nous renvoyons le lecteur à Ehrgott et Gandibleux [Ehrgott et al.02].

TGV que de trains de marchandises si le TGV rapporte plus,
- la stabilité de l'horaire, c'est à dire diminuer au mieux l'impact du retard des trains sur les autres. De cette façon, si un train a du retard son impact sera limité sur les autres,
- etc ...

Les résultats obtenus sur des instances réelles et aléatoires ont prouvé l'efficacité des algorithmes heuristiques et l'utilité des pré-traitements (résultats de bonne qualité en un temps de réponse d'une demi heure) sur ce type de problème. Ces résultats ont également mis en évidence les limites de la résolution exacte.

1.4.5 COMBINE

Les projets COMBINE [Mascis et al.04], [Paciarelli et al.06] et COMBINE 2 [Paciarelli et al.02] (COntrol center for a Moving Block sIgNalling systEm) ont pour objectif l'optimisation du trafic ferroviaire en temps réel, ainsi que l'évaluation de la faisabilité de grille horaire établie. Ces projets exploitent l'infrastructure équipée de cantons mobiles, c'est à dire que les cantons de l'infrastructure ne sont plus fixes mais évoluent avec le train. Ce type de cantonnement n'est possible que par des équipements de signalisation adaptés et différents de ceux présentés précédemment (section 1.1). En effet, les limites du canton qui protègent un train ne peuvent plus être statiques et implantées sur la voie mais doivent être dynamiques et envoyées en temps réel aux trains environnants afin de respecter le principe d'espacement. Le standard européen qui intègrera ce type de signalisation est le système ERTMS (European Railway Traffic Management System)[Group99] :"Le système ERTMS a pour objectif d'apporter au niveau des systèmes de signalisation, une solution économique et technique à l'interopérabilité ferroviaire, rendue indispensable par l'accroissement des échanges au sein de l'Union Européenne". Suivant le type de ligne et les performances requises (trafic, vitesse), quatre modes de signalisation peuvent être utilisés :
- Niveau 1 : Système ATP (Protection Automatique de Train) ponctuel à balise,
- Niveau 2 : Système ATP continu par radio GSM,
- Niveau 3 : Système ATP continu par radio GSM avec localisation du train par lui-même,
- Niveau 4 : Système ATP semi-continu par radio pour les lignes secondaires à faible débit.

COMBINE se base sur une signalisation ERTMS de niveau 3, COMBINE 2 quant à lui étend cette signalisation en vue d'optimiser la capacité du trafic sur un large réseau.

La modélisation est effectuée par un "graphe alternatif". Ce type de représentation reprend un peu le concept d'un réseau de Petri. Un train est une séquence de tâches correspondant à l'occupation de chaque canton de son parcours. Dans le graphe, les nœuds sont les dates de début d'entrée d'un train dans un canton. Un premier type d'arc relie les noeuds du parcours d'un train, ces arcs sont étiquetés par la durée d'occupation du canton et correspondent à des contraintes potentielles comme dans le modèle CAPRES (voir section 1.4.2).

Un autre type d'arc, appelé "arc alternatif", permet de bloquer la transition vers le nœud suivant tant que certaines conditions ne sont pas respectées (fin d'utilisation d'un autre nœud notamment).

Ce type d'arc permet de représenter les contraintes disjonctives entre deux tâches pour l'accès à une ressource (le canton) comme dans le modèle CAPRES (voir section 1.4.2).

La figure 1.8 illustre un exemple de graphe alternatif entre les tâches i et j de deux trains différents qui requièrent un même canton. Dans cet exemple, la tâche suivant la tâche i (respectivement j) est notée $s(i)$ (respectivement $s(j)$). Les arcs (en traits pleins dans la figure 1.8) qui relient les noeuds des débuts des tâches successives d'un même train sont étiquetés t_i et t_j. Les valeurs t_i, t_j correspondent aux durées de traversée du canton pour chaque train. Chaque arc alternatif (en pointillés dans la figure 1.8) réprésente un ordre différent d'occupation du canton, donc de passage des trains. Les deux séquencements possibles sont :

- $i \prec j$: l'arc $(s(i), j)$ indique que le début de la tâche j sera postérieur ou égal au début de $s(i)$ avec un délai de q_j.
- $j \prec i$: l'arc $(s(j), i)$ indique que le début de la tâche i sera postérieur ou égal au début de $s(j)$ avec un délai de q_i.

La valeur q_i (respectivement q_j) est la somme d'une marge de sécurité constante et de la durée du dégagement du canton par le train après la fin de la tâche j (respectivement i).

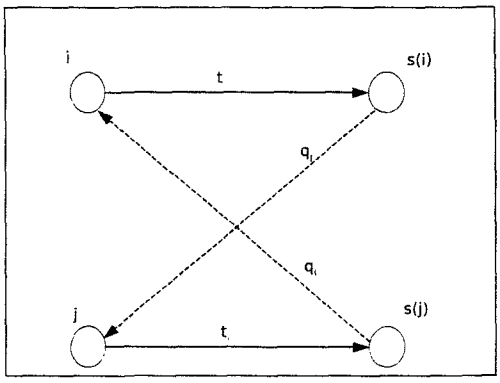

FIG. 1.8 – Représentation d'une contrainte de blocage par un graphe alternatif

L'algorithme d'ordonnancement minimise une fonction des retards de sortie agissant au niveau des contraintes de précédence entre trains, et au niveau des routages des trains. Les principales tâches de l'algorithme pour créer le plan de transport optimisé suivant la grille horaire, les contraintes possibles et la positions des trains, sont :

1. génération du graphe alternatif,
2. prétraitement sur le graphe : trouver les relations de précédences obligatoires (déduites des relations de précédences fixées),
3. vérification du graphe : vérification qu'il ne contient pas de cycle négatif.
4. détection des conflits,
5. résolution des conflits et propagation sur le choix effectué,

6. si le graphe est cyclique, alors il n'y a pas de solution, sinon il reste des conflits aller en 4.

Si l'algorithme finit avec une solution réalisable il est possible d'en rechercher une meilleure grâce à un re-routage des trains. Le principe est de sélectionner un train et de tester sa meilleure alternative (meilleur re-routage possible pour ce train) et ce pour chaque train, tant que le critère d'arrêt n'est pas atteint (limite de temps, qualité, ...). Les trains sélectionnés en priorité sont ceux ayant les arcs les plus critiques (un arc impliquant une importante accumulation de retards sur les autres). Cet algorithme utilise les algorithmes issus de la théorie des graphes et un Branch & Bound tronqué (algorithme glouton avec quelques retours arrière possible). À chaque décision prise, le graphe alternatif est mis à jour en utilisant des techniques d'ordonnancement (edge finding [Caseau et al.94]).

L'utilisation de la capacité offerte par l'infrastructure étudiée est effectuée par l'algorithme présenté précédemment. Cet algorithme génère, à partir de la grille horaire donnée, l'ordonnancement optimum. Des tests (cf. [Rivier et al.01]) ont été effectués sur la faisabilité d'une grille horaire saturée d'une infrastructure impliquant une modélisation à 1600 nœuds, 1700 arcs fixés et 7400 arcs alternatifs. Les résultats obtenus indiquent que la densité souhaitée est réalisable et que l'algorithme permet, par rapport à un algorithme de régulation de trafic conventionnel, de :

- diminuer les temps de parcours,
- prendre en compte les aléas plus aisément,
- diminuer la consommation d'énergie (régulation de la vitesse),
- contrôler plus efficacement le trafic global.

Il est également à noter que le projet COMBINE comporte une partie importante sur la régulation de la vitesse des trains sur l'infrastructure.

1.5 Synthèse des travaux présentés

L'ensemble des projets décrits considère l'optimisation de la capacité ferroviaire, les modélisations proposées sont assez proches pour les quatre premiers projets (CAPRES pouvant également se démarquer par l'utilisation des techniques de programmation par contraintes). COMBINE présente une approche originale et permet de prendre en compte les nouveaux types de signalisations tel que ERTMS. Le tableau 1.2 résume les principales caractéristiques de ces projets.

Si ces projets considèrent tous le problème de la capacité d'infrastructure ferroviaire, ils ne répondent pas aux mêmes questions. En effet, ils répondent tous au problème de faisabilité, mais les problèmes de saturation, d'optimisation des préférences et de fluidification ne sont pas toujours pris en compte.

La difficulté de la résolution de ces problèmes se traduit par la diversité des méthodes proposées. Les méthodes de propagations de contraintes ou basées sur les ajouts de coupes sont plus avantageuses car ces problèmes sont fortement contraints. Les temps de réponse de ces méthodes,

Projet	DONS	CAPRES	DÉMIURGUE	RECIFE	COMBINE
Infrastructure	Réseau (CADANS) Nœud (STATIONS)	Réseau	Réseau	Nœud	Réseau
Signalisation	fixe	fixe	fixe	fixe	Mobile
Résolution	PPC, Branch & Bound, méthodes de coupes	PPC	Branch & Cut	Branch & cut GRASP	Branch & Bound, PPC

TAB. 1.2 – Synthèse des différentes études de capacité

ainsi que l'applicabilité des résultats (mise en œuvre réelle des grilles horaires construites) dépendent fortement de la précision des données.

La problématique posée pour cette thèse est la résolution du problème de saturation au niveau d'un nœud, ce thème correspond à celui du projet RECIFE étudié par Delorme [Delorme03]. L'hypothèse du projet RECIFE est un intervalle de temps fixe pour la circulation des trains. La modélisation proposée dans ce projet ne permet pas un lien direct entre les variables de décisions et la réalité physique (correspondance variables de décision / trains). De plus, les difficultés rencontrées lors de la résolution exacte sur les instances proches de la réalité, il est intéressant de tester une nouvelle méthode de résolution afin de pouvoir mieux évaluer et même améliorer les résultats obtenus. Ainsi cette thèse prend comme hypothèse un ensemble de trains fixés et non pas un intervalle de temps. Une nouvelle modélisation et de nouveaux algorithmes de résolution pour le problème de saturation d'infrastructures ferroviaires au niveau d'un nœud sont proposés. Les résultats seront évalués en référence avec ceux obtenus par Delorme dans [Delorme03]. Ce travail fait partie intégrante du projet RECIFE II, projet prenant la suite du projet RECIFE.

1.6 Conclusion

Ce chapitre a présenté l'infrastructure ferroviaire, principale ressource du problème traité. Les différents types de conflits entre trains et les principes permettant de les gérer ont été présentés.

Le problème traité a été positionné par rapport aux niveaux de décisions des modes de transport. L'intérêt du problème a été argumenté par l'augmentation du trafic ferroviaire à venir, ce qui demande une utilisation plus efficace des infrastructures existantes.

La définition du problème de l'évaluation de la capacité d'infrastructure ferroviaire a été donnée et expliquée. Les différents problèmes émanant de cette évaluation, notamment le problème de saturation, ont été détaillés.

Les principaux travaux portant sur la capacité ferroviaire ont été décrits. Leurs avantages et inconvénients ont été rapportés.

Le travail de cette thèse, placé dans le cadre d'un projet industriel RECIFE II, projet prenant la

suite du projet RECIFE, a été motivé. En effet le transport ferroviaire possède une grande part de planification. L'évaluation de la capacité fait partie de la planification à long terme car elle permet d'évaluer l'infrastructure sur la question : L'infrastructure actuelle peut elle ou non absorber le trafic futur ? Dans le cas négatif, quels aménagements semblent être les plus adéquats ? Dans le contexte actuel, la demande est de plus en plus forte, aussi les infrastructures doivent être adaptées, leur utilisation optimisée et ce dans des temps de calcul acceptables pour l'utilisateur.

La suite présente les outils utilisés pour traiter ce problème ainsi que sa modélisation, sa résolution et les résultats obtenus sur des instances ferroviaires réelles et aléatoires.

Chapitre 2

Généralités sur la programmation par contraintes

Un problème d'optimisation demande, en général, une bonne solution (voire optimale) ou la preuve qu'il n'en existe pas. La question est alors de savoir comment trouver cette ou ces solution(s), voire la meilleure. Pour ce faire des techniques de satisfaction de contraintes et d'optimisation ont été développées.

L'optimisation a pour objectif de trouver la ou les solutions optimales du problème. Une solution optimale peut être vue comme une solution répondant à un ensemble de directives permettant de réaliser une tâche au mieux suivant un critère donné. Les problèmes sont décrits par des variables de décision sur lesquelles des caractéristiques (directives), telles que des contraintes, doivent être vérifiées. Le but est de trouver les valeurs des variables répondant aux caractéristiques tout en donnant la valeur minimale (ou maximale) du critère choisi. Cependant la résolution de ces problèmes est souvent impossible manuellement. En effet, le nombre de paramètres, de contraintes, de variables rend la construction d'une solution souvent délicate et plus encore la construction de la solution optimale.

Ce chapitre présente la modélisation à base de contraintes et les différentes techniques de résolution associées utilisées dans le cadre de cette thèse. Le chapitre est organisé en sept parties. Tout d'abord les bases des problèmes de satisfaction de contraintes (définition, propagation, techniques générales de résolution) sont présentées dans les sections 2.1, 2.2, 2.3, 2.4. Ensuite, les problèmes de satisfaction de contraintes temporelles (définition, propagation) sont décrits dans la section 2.5. Enfin, les sections 2.6 et 2.7 sont consacrées aux techniques de résolution des heuristiques et métaheuristiques courantes.

2.1 Problèmes de Satisfaction de Contraintes

Dans le domaine de la modélisation de problèmes, les relations entre les variables sont exprimées à l'aide de contraintes. Une contrainte constitue un élément de représentation de la connaissance. La programmation par contraintes est un outil informatique permettant à la fois d'exprimer les contraintes qui décrivent un problème et de résoudre ces problèmes par des algorithmes. Elle distingue et allie deux fonctionnalités :

1. La représentation des connaissances : les problèmes sont décrits à l'aide de variables, chacune associée à un ensemble de valeurs possibles appelé domaine et de contraintes qui expriment des relations entre les variables,
2. La résolution de problèmes : le processus de résolution, basé sur l'exploration limitée de l'espace des solutions du problème à traiter, alterne la construction d'un arbre de recherche et la propagation (élimination de valeurs incohérentes dans le domaine des variables) des contraintes à chaque nœud de l'arbre.

Les techniques de satisfaction de contraintes permettent non seulement de formuler et de résoudre des problèmes mais également de diminuer l'espace de recherche en considérant la structure du problème, de guider la recherche vers la ou les solution(s). Cette théorie fournit un cadre simple et formel, dans lequel des techniques de raisonnement automatique sont développées.

Définition 7 (Problème de Satisfaction de Contraintes [Montanari74], [Mackworth77]). *Un Problème de Satisfaction de Contraintes ou CSP est défini par la donnée d'un triplet (X, D, C) où :*

- $X = \{x_1, \ldots, x_n\}$ *est un ensemble fini de variables,*
- $D = \{d_1, \ldots, d_n\}$ *est l'ensemble fini des domaines de valeurs des variables de X. À chaque variable x_i est associée son domaine de valeurs d_i.*
- $C = \{c_1, \ldots, c_m\}$ *est un ensemble fini de contraintes, chaque contrainte porte sur un sous-ensemble de variables de X. $c_i(x_{i1}, \ldots, x_{ik})$ représente en extension un sous-ensemble du produit cartésien $d_{i1} \otimes \ldots \otimes d_{ik}$ correspondant aux valeurs des variables de $\{x_{i1}, ..., x_{ik}\}$ compatibles entre elles.*

Une solution d'un CSP est une affectation de l'ensemble des variables du problème qui satisfait toutes les contraintes. Lorsqu'une instance (ensemble de contraintes, domaines) du CSP a été définie, on peut vouloir trouver une solution ou toutes les solutions du problème.

En programmation par contraintes, un problème d'optimisation combinatoire est défini par la combinaison d'un CSP et d'une fonction objectif. La mise en œuvre de la recherche d'un optimum se traduit dans l'algorithme de recherche par l'ajout d'une contrainte après l'obtention d'une solution du CSP. Cette contrainte ajoutée oblige l'algorithme de recherche à trouver une meilleure solution que la meilleure trouvée précédemment.

2.2 Propagation de contraintes

Le filtrage des domaines (propagation de contraintes) est couramment utilisé comme prétraitement de chaque étape de la résolution d'un CSP. Ceci permet de réduire les domaines des variables et de simplifier la résolution d'un problème ou de démontrer l'absence de solutions. Ainsi la propagation de contraintes désigne un ensemble de techniques de réduction de l'espace de recherche ne modifiant pas l'ensemble des solutions.

2.2.1 Notion de consistance

Dans cette section les définitions d'instanciation partielle localement et globalement consistante (d'après [Dechter92]) sont données :

Définition 8 (Consistance locale). *Par rapport à un $CSP(X, D, C)$, une instanciation partielle (affectation de valeurs à un ensemble de variables Y tel que $Y \subseteq X$) est dite localement consistante si elle satisfait toute contrainte $c_i(x_{i1}, ..., x_{ik})$ telle que $\{x_{i1}, ..., x_{ik}\} \subseteq Y$.*

Définition 9 (Consistance globale (1)). *Une solution du $CSP(X, D, C)$ est une instanciation de X localement consistante. Cette instanciation est dite globalement consistante.*

Définition 10 (Consistance globale (2)). *Une instanciation partielle est globalement consistante si elle peut être étendue à une solution.*

2.2.2 Propriétés des algorithmes de filtrage

Le filtrage consiste, comme décrit dans [Gensel95], à réduire le $CSP(X, D, C)$ d'origine en un $CSP(X, D', C')$ équivalent, c'est à dire portant sur le même ensemble de solutions et tel que le domaine D' de toute variable de X soit inclus ou égal à son domaine de départ. Un algorithme de filtrage est dit :

- sain si suite au filtrage aucune valeur consistante n'a été supprimée,
- complet si suite à son application aucune valeur inconsistante ne peut être retirée.

2.2.3 Nœud-consistance

Elle ne s'applique qu'aux contraintes unaires (portant sur une seule variable) et ne permet pas, en général, de diminuer de façon significative le domaine des variables.

Définition 11 ([Mackworth77]). *Un CSP est nœud-consistant si et seulement si :*
$\forall x_i \in X, \forall v_i \in d_i,$
$\forall c_k \in C,$ *telle que* c_k *ne porte que sur* x_i
v_i $satisfait$ c_k

2.2.4 Arc-consistance

L'ensemble des contraintes qui relie les variables forme le graphe des contraintes où les nœuds sont les variables et les arcs les contraintes, d'où la dénomination d'arc-consistance. Il s'agit de la consistance la plus utilisée pour les CSP binaires. Un CSP binaire est un CSP dont les contraintes ne portent que sur deux variables. On désignera par c_{ij} la contrainte binaire liant la variable x_i à x_j.

Définition 12 ([Mackworth77]). *Un CSP est arc-consistant si et seulement si :*
$\forall x_i \in X, \forall v_i \in d_i,$
$\forall c_k\{x_{k1}, \ldots, x_{kp}\} \in C$, *telle que* $x_i \in \{x_{k1}, \ldots, x_{kp}\}$
$\exists v_{k1} \in d_{k1}, \ldots, \exists v_{kp} \in d_{kp}$, *telles que* $(v_{k1}, \ldots, v_i, \ldots, v_{kp}) \in c_k$

Le but des algorithmes d'arc-consistance est de propager le fait qu'une (ou plusieurs) valeur(s) v_i de la variable x_i n'aient pas de valeurs admissibles dans le domaine de la variable x_j pour satisfaire la contraintes c_{ij} (l'instanciation (v_i, v_j) n'est pas localement consistante). On élimine cette valeur du domaine d_i et l'on propage cette suppression aux autres variables liées à x_i par une contrainte. Un exemple de graphe arc-consistant est présenté en figure 2.1.

Il existe, à ce jour, 7 versions d'algorithmes permettant d'établir l'arc-consistance (AC-1 à AC-7) se distinguant par leur finesse de choix. Le plus utilisée est AC-3 [Mackworth77], décrit par l'algorithme 2.2.1.

algorithme 2.2.1 AC-3

Procédure AC-3(X,D,C)
début
  $Q \leftarrow \{(i,j) | c_{ij} \in C\}$
  ! arcs issus des contraintes de C !
  tantque $Q \neq \emptyset$ **faire**
    enlever (i,j) de Q
    si Révise((i,j),X,D,C) **alors**
      $Q \leftarrow Q \cup \{(k,i) \in C \wedge k \neq i \wedge k \neq j\}$
    finsi
  fintantque
fin

Fonction Révise$((i,j), X, D, C)$
début
  Révise $\leftarrow$ faux
  pour tout $v_i \in d_i$ **faire**
    si $\nexists v_j \in d_j$ tel que c_{ij} soit satisfaite **alors**
      $d_i \leftarrow d_i - \{v_i\}$
      Révise $\leftarrow$ vrai
    finsi
  finpour
fin

2.2.5 Consistance de chemin

La consistance de chemin permet d'établir la consistance entre les valeurs de deux variables liées par un ensemble de contraintes binaires. La consistance de chemin vérifie que toute paire de valeurs des variables est arc-consistante et qu'en plus il existe une suite de valeurs compatibles deux à deux dans tous les chemins qui relient les deux variables dans le graphe des contraintes. Elle

fut introduite par Mackworth et [Mackworth77] Montanari en 1974 [Montanari74] de la manière suivante :

Définition 13 ([Mackworth77]). *Un CSP binaire (les contraintes ne portent que sur 2 variables) est* ***chemin-consistant*** *si et seulement si :*
Quelque soit le chemin $(x_0, x_1, ..., x_k)$ *dans le graphe où les arêtes sont des contraintes binaires,* $\forall v_0 \in d_0, \ldots, \forall v_k \in d_k$,
l'instanciation de deux valeurs $(x_0 = v_0, x_k = v_k)$ *est localement consistante et* $\forall p \in [1, k-1], \exists v_p \in d_p$ *telle que toute instanciation de deux valeurs* $(x_{p-1} = v_{p-1}, x_p = v_p)$ *soit localement consistante.*

La consistance de chemins pour un problème à contraintes binaires peut être obtenue de façon efficace (introduite par Montanari [Montanari74]). Elle correspond à la 3-consistance du formalisme de Freuder [Freuder82] et Montanari [Montanari74].

La figure 2.1 présente un exemple de CSP arc-consistant mais non chemin-consistant. Sur ce graphe, les variables sont représentées par les nœuds, les contraintes par les arcs, la notation $\{a, b\}$ correspond au domaine de la variable, la notation (a, b) indique un tuple de valeur admissible pour le couple de variable relié par l'arc.

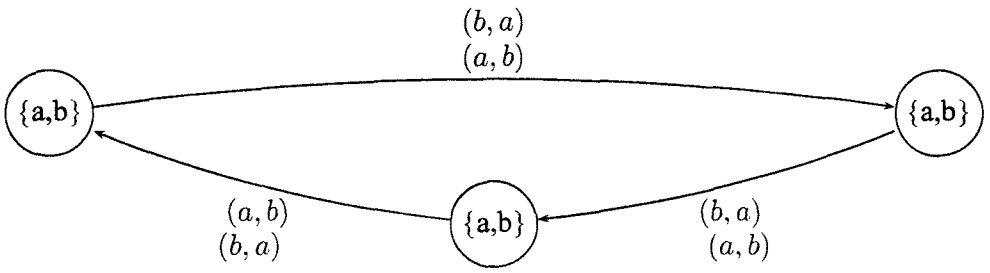

FIG. 2.1 – CSP arc-consistant mais non chemin consistant

La section suivante présente donc la k-consistance et le lien avec la consistance de chemin.

2.2.6 k-consistance

La k-consistance est une définition généralisée des divers degrés de consistance atteignables. Par exemple, l'arc-consitance équivaut à la 2-consistance. La k-consistance revient à considérer si toute instanciation consistante de k-1 variables peut être étendue à une instanciation de k variables. Plus k est grand, plus le filtrage sera efficace, mais plus la résolution sera lourde. Cette consistance fut introduite par Freuder [Freuder82] :

Définition 14 (Freuder, 1982). *Un CSP est k-consistant* $\forall k \in [1, n]$ *si et seulement si :*
$\forall (x_1 = v_1, \ldots, x_{k-1} = v_{k-1})$, *instanciation partielle localement consistante de* $k-1$ *variables de* X,

$\forall x_k \in X$ *telle que* $x_k \notin \{x_1, \ldots, x_{k-1}\}$,
$\exists v_k \in d_k$ *telle que* $(x_1 = v_1, \ldots, x_{k-1} = v_{k-1}, x_k = v_k)$ *soit localement consistante.*

Pour établir la consistance de chemin, la 3-consistance est suffisante. Un réseau de contraintes vérifie la 3-consistance si et seulement si la propriété suivante est respectée :

$$\forall i, j, k \in [1, n], \forall (v_i, v_j) \in C_{ij}, \exists v_k \in D_k | (v_i, v_k) \in C_{ik} \text{ et } (v_k, v_j) \in C_{kj} \quad (2.1)$$

Pour établir la 3-consistance, il suffit d'enlever toutes les valeurs v_i appartenant au domaine D_i qui n'appartiennent à aucun couple (v_i, v_j) vérifiant la propriété ci-dessus.

La complexité des algorithmes permettant d'obtenir la k-consistance est en $O(n^k \times d^k)$, où n représente le nombre de variables et d la taille du plus grand domaine.

2.3 Analyse et suppression des symétries

2.3.1 Définition

Les symétries divisent l'ensemble des affectations possibles d'un CSP donné en classes d'équivalences. Le passage d'une classe à l'autre s'effectue par une fonction de transformation des affectations (différente de l'identité). Chaque classe d'équivalence contient ou non des solutions. Beaucoup de problèmes de satisfaction de contraintes et d'optimisation contiennent des symétries, ce qui implique des solutions équivalentes. Lors de la recherche de solutions d'un CSP, seules les solutions d'une classe sont nécessaires, les autres pouvant être obtenues en utilisant ces symétries (c'est à dire la fonction de transformation). Soit δ^σ l'application de la symétrie σ à la décision δ :

Définition 15 (Symétrie [Puget05]). *Une symétrie σ pour un CSP $\mathcal{P}$ est une relation d'équivalence entre les décisions prises sur $\mathcal{P}$ sujet à :*

- *L'application de la symétrie σ à une affectation $A = \{x_i = v_i\}$ correspond à l'affectation $A^\sigma = \{(x_i = v_i)^\sigma\}$.*
- *Pour toute affectation A, $A \in sol(\mathcal{P})$ si et seulement si $A^\sigma \in sol(\mathcal{P})$, avec $sol(\mathcal{P})$ représentant l'ensemble des solutions du problème $\mathcal{P}$.*

2.3.2 Techniques de suppression des symétries

Quand une seule solution est demandée, la recherche se limite à un seul élément de chaque classe, afin d'éviter les recherches redondantes. Les symétries ont fait l'objet de nombreuses recherches afin de les détecter et de les éliminer. Plusieurs méthodes ont ainsi été proposées :

- ajout de contraintes sur le modèle [Crawford et al.96],
- ajout de contraintes sur le sous-arbre de recherche courant [Puget03],
- utilisation d'heuristiques cassant les symétries [Meseguer et al.01],

- utilisation de techniques de recherche éliminant les valeurs interchangeables [Degoutin et al.05a],
- utilisation de sous-arbres de recherche complets comme "no-good" (assignement de variables ne pouvant pas donner de solution, cf. [Schiex et al.94] et [Dechter90]) afin d'éviter l'exploration de sous-arbres symétriquement équivalents. Soit des contraintes sont ajoutées durant la recherche [Backofen et al.99], soit une détection de dominance, durant la recherche, est effectuée [Focacci et al.01b], [Puget05].

Le modèle ci-dessous illustre le cas d'un problème symétrique.

Exemple Symétries
Soit un problème $\mathcal{P}$ défini comme suit :

$$\begin{cases} \min z(x) = x_1 + x_2 \\ s.c \\ \quad x_1 + x_2 = 1, \\ \quad x_i \in \{0;1\} \end{cases} \tag{2.2}$$

Ce problème est bien symétrique car la solution $x_1 = 0, x_2 = 1$ est totalement équivalente à la solution $x_1 = 1, x_2 = 0$, les valeurs de ces variables ayant été interchangées et la valeur du critère n'a pas été modifié. Afin d'éliminer ce type de symétrie, la contrainte $x_i > x_j, \forall i < j$ peut être ajoutée au problème.
Un autre exemple de problème symétrique est le jeu du Sudoku [], où, sous certaines conditions, l'échange de lignes ou colonnes peut donner une autre solution.

2.4 Techniques de résolution

Les techniques de résolution en Programmation Par Contraintes (PPC) s'appuient principalement sur un algorithme par instanciation de variable et retour arrière en cas d'échec avec différents types de sélection de variables et valeurs à instancier.

2.4.1 Stratégies de recherche de solutions

Dans la recherche d'une solution, on distingue deux types d'opérations : d'une part l'instanciation d'une variable et, d'autre part, le test de la validité des instanciations partielles réalisées jusqu'alors.

Les méthodes de recherche de solutions se divisent en deux classes :

1. **les méthodes *rétrospectives*** qui, en cas d'échec, remettent en cause une variable précédemment instanciée. Les principaux algorithmes qui utilisent cette approche sont :

- *le backtracking* où la variable dont l'instanciation remise en cause est la précédente dans la chronologie d'instanciation,
- *le backjumping [Dechter90]* où l'on tient compte d'abord de la structure du problème en considérant les variables partageant une contrainte avec la dernière variable instanciée puis le critère chronologique,
- *le backchecking [Haralick et al.80]* consiste, par exemple, à se rappeler qu'une valeur v_i choisie pour la variable x_i en cours d'instanciation est incompatible avec la valeur v_j de la variable x_j. Tant que x_j aura pour valeur v_j, x_i ne sera pas instanciée à v_i.
- *le backmarking [Gaschnig77]* qui, pour un niveau d'instanciation d (d représente ici l'indice de la variable instanciée et le niveau de l'arbre de recherche), enregistre le plus haut niveau dans l'arbre de recherche vers lequel un retour arrière doit être opéré, ainsi que le plus bas niveau pour lequel v_d (valeur associée à la variable d) est compatible avec toutes les variables déjà instanciées.

2. **les approches prospectives ou *Lookahead* [Haralick et al.80]**, l'idée est de vérifier qu'à chaque instanciation, il existe des valeurs compatibles dans les domaines des variables non encore instanciées. Ces approches se distinguent par l'intensité du filtrage qu'elles opèrent. Deux algorithmes sont principalement utilisés :
   - l'algorithme *Forward-Checking* élimine des domaines des variables non instanciées, les valeurs incompatibles avec la valeur de la variable en cours d'instanciation. Lorsqu'un domaine d'une variable non instanciée est vidé et qu'il ne reste plus de valeur candidate pour la variable en cours d'instanciation, le Forward-Checking remet en cause la valeur de la variable précédemment instanciée.
   - le *Full-Lookahead* consiste en l'établissement de l'arc-consistance du réseau formé par les variables non encore instanciées.

   Plus le filtrage utilisé est puissant, plus le nombre de tests est grand. En revanche, l'espace de recherche est diminué et les filtrages suivants porteront sur des domaines réduits. Il faut donc trouver un bon compromis entre la puissance du filtrage appliqué à chaque noeud de l'arbre de recherche et le nombre de noeuds à explorer.

2.4.2 Heuristiques d'ordre

L'ordre d'instanciation des variables, tout comme l'ordre des valeurs choisies, peut avoir des répercussions sur les performances de la résolution. On parle d'heuristiques car il n'est pas sûr que le critère choisi aboutisse à un arbre de recherche de taille minimum quelque soit le CSP considéré.

1. **Ordre sur les variables**

   Ces heuristiques utilisent l'un des critères suivants :
   - la largeur minimale (instancier les variables les plus contraintes et au domaine le plus faible d'abord),

- la cardinalité minimale (choix arbitraire de la première variable puis choix de la variable suivante partageant le plus de contraintes avec la précédente),
- le degré maximal (classement de la variable la plus contrainte à la moins contrainte).

On peut aussi citer :

- le *first fail principle [Haralick et al.80]* : cette heuristique se fonde sur le principe indiquant qu'"Afin de réussir, essayer d'abord où la probabilité d'échec est la plus grande", de cette façon les échecs des instanciations les plus difficiles permettront de couper l'arbre de recherche plus tôt et d'en réduire sa taille.
- le réarrangement dynamique : il s'agit de sélectionner la meilleure variable (du point de vue de l'heuristique choisie) pour le nœud courant.

2. **Ordre sur les valeurs**

   Une fois la variable à instancier choisie, l'ordre d'instanciation des valeurs du domaine peut aussi contribuer à l'efficacité de la résolution.

   Un des ordres les plus pratiqués est la méthode des *conflits minimaux*. En partant d'une instanciation totale de l'ensemble des variables, une des variables en conflit (variable impliquant la non satisfaction d'une contrainte) est choisie et affectée à une valeur permettant de diminuer le nombre de conflits de cette variable avec les autres.

   [Haralick et al.80], [Purdom83] ou [Dechter et al.94] donnent une étude plus précise sur l'ordre d'instanciation des valeurs.

2.4.3 Résumé

Les principales techniques de la théorie des problèmes de satisfaction de contraintes viennent d'être présentées. L'arc-consistance et l'élimination des symétries sont des moyens de réduire l'espace de recherche des solutions. Les techniques de filtrage par consistance tout comme certaines heuristiques d'ordre nécessitent des temps de calcul, il faut alors trouver un bon compromis entre réduction de la taille de l'arbre de recherche et temps de calcul. Le recours à un ordonnancement des variables ou valeurs est souvent bénéfique. Les techniques qui ont été présentées s'appliquent à tous les types de CSP. Pour certains, des techniques spécifiques ont été développées, c'est le cas des CSP temporels qui font l'objet de la section suivante.

2.5 Les CSP temporels

Beaucoup de problèmes, notamment les problèmes d'ordonnancement [1], possèdent des contraintes temporelles. Il s'agit d'une classe particulière des CSP où les variables correspondent à des dates

[1] Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises

d'événements et les contraintes à des relations entre ces événements. Les domaines des variables ne sont donc plus des ensembles possibles de valeurs discrètes mais des intervalles de valeurs.

Pour une bibliographie détaillée sur les problèmes de satisfaction de contraintes temporelles, le lecteur est renvoyé à [Dechter et al.91], [Gensel95] et [Esquirol et al.01].

2.5.1 CSP à intervalles (ICSP)

Le domaine associé à chaque variable est un intervalle de réels. La variable associée à ce domaine peut donc prendre une infinité de valeurs. L'application de l'algorithme d'arc-consistance n'est alors plus possible étant donné qu'il faudrait pouvoir tester chaque valeur du domaine d'une variable avec toutes les valeurs possibles des variables liées à elle par une contrainte.

On peut aussi avoir des ICSP (CSP à intervalles) limités aux entiers, dans ce cas établir l'arc-consistance est possible et efficace si les intervalles considérés ne sont pas trop grands.

Plusieurs degrés de consistance ont alors été définis dans [Benhamou et al.94] : l'*intervalle-consistance*, l'*enveloppe-consistance*, la *boîte-consistance* et l'*arc-consistance de bornes*. L'arc-consistance de bornes (ou arc-B-consistance) est la plus couramment utilisée par les solvers commerciaux, aussi elle est définie ici pour un ICSP (CSP à intervalles) :

Définition 16 (CSP à Intervalles [Lhomme93]). *Un ICSP(X,D,C) est arc-B-consistant si et seulement si*

$\forall x \in X$ *telle que* $d_x = [a, b], \forall C(x, x_1, ..., x_k) \in C$, *implique :*
$\exists v_1, ..., v_k \in d_1 \times ... \times d_k$ *telles que* $C(a, v_1, ..., v_k)$ *est satisfaite*
$\exists v_1', ..., v_k' \in d_1 \times ... \times d_k$ *telles que* $C(b, v_1', ..., v_k')$ *est satisfaite.*

L'arc-consistance de bornes est donc une consistance limitée aux bornes des intervalles des variables.

2.5.2 Problèmes de satisfaction de contraintes temporelles (TCSP)

Un problème de satisfaction de contraintes temporelles (TCSP) est défini par :

- un ensemble de variables temporelles $\{x_1, \ldots, x_n\}$ correspondant à des instants,
- un ensemble de domaines associés à chaque variable $\{d_1, \ldots, d_n\}$. Chaque domaine d_i représente l'ensemble des valeurs pouvant être prises par x_i ; chaque domaine d_i est l'union d'un ensemble de n_i intervalles disjoints : $d_i = I_i^1 \cup I_i^2 \cup \ldots \cup I_i^{n_i}$,
- un ensemble de contraintes binaires C, c_{ij} limitant la distance entre 2 variables, $x_j - x_i$, par un domaine d_{ij} de n_{ij} intervalles disjoints. c_{ij} représente la contrainte $x_j - x_i \in d_{ij}$.

Les contraintes de domaines peuvent être mises sous forme de contraintes binaires de la forme c_{0i} ou c_{i0} avec la variable x_0 représentant l'origine.

Deux opérations principales sur les contraintes temporelles peuvent être définies :

- l'opération d'*intersection*, notée $\oplus$, est définie entre deux contraintes binaires portant sur les mêmes variables. Cette opération permet de rassembler les deux contraintes en une seule. L'intersection des deux contraintes $C_{ij}^1 : x_j - x_i \in d_{ij}^1$ et $C_{ij}^2 : x_j - x_i \in d_{ij}^2$ correspond à la contrainte $C_{ij}^3 = C_{ij}^1 \oplus C_{ij}^2$, soit $x_j - x_i \in d_{ij}^3$ avec $d_{ij}^3 = d_{ij}^1 \cap d_{ij}^2$;
- l'opération de *composition*, notée $\otimes$, est définie entre deux contraintes possédant une variable commune. Cette opération permet d'établir la contrainte redondante liant deux variables (i et j) par l'intermédiaire d'une troisième (k), comme décrit sur la figure 2.2. La composition de deux contraintes $C_{ik} : x_k - x_i \in d_{ik}$ et $C_{kj} : x_j - x_k \in d_{kj}$ correspond à la contrainte $C_{ij} = C_{ik} \otimes C_{kj}$, soit $x_j - x_i \in d_{ij}$ avec $d_{ij} = \{ v | \exists v_1 \in d_{ik}, \exists v_2 \in d_{kj}, v = v_1 + v_2 \}$

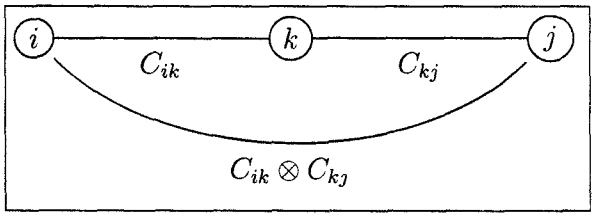

FIG. 2.2 – Principe de la composition

Un exemple de l'intersection et de la composition de 2 intervalles : I et J, est illustré sur la figure 2.3. Sur l'exemple de l'opération de composition, l'intervalle I (respectivement J) considéré correspond à l'intervalle de la contrainte C_{ik} (respectivement C_{kj}).

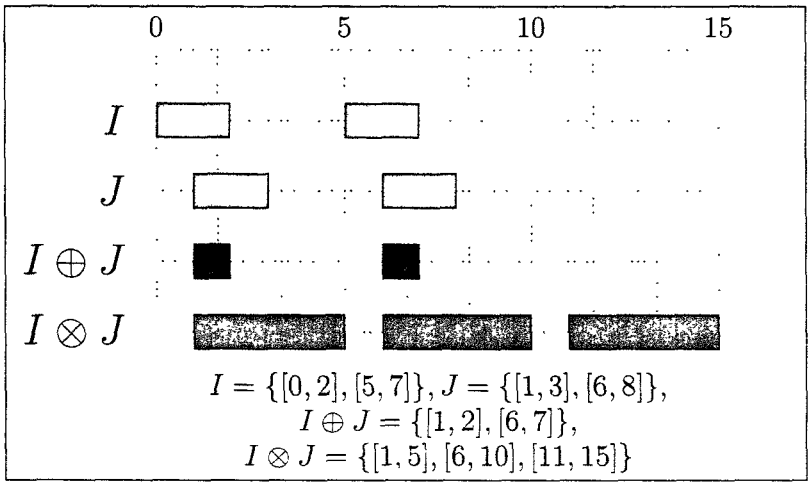

FIG. 2.3 – Opérations d'intersection et de composition

À partir de ces définitions générales sur les TCSP, on dérive une classe particulière de problèmes dans lesquels toutes les contraintes ne font intervenir qu'un seul intervalle de valeurs. Cette classe possédant une propriété très intéressante, détaillée en section suivante le montrera, nous verrons ensuite différents algorithmes de propagations spécifiques aux TCSP.

2.5.3 Propagation pour les problèmes temporels simples (STP)

Pour ce type de CSP, l'ensemble des contraintes temporelles se ramène à un ensemble d'inégalités linéaires entre deux variables, autrement dit à des contraintes du type :

$$a_{ij} \leq x_j - x_i \leq b_{ij}$$

Pour un problème de satisfaction de contraintes temporelles simple STP, l'algorithme de Floyd-Warshall (2.5.1) établit la 3-consistance. Montanari [Montanari74] montre, pour les STP, que la 3-consistance est équivalente à la consistance de chemins. Cet algorithme, appelé aussi PC-0, permet de déterminer le domaine minimal de toute contrainte C_{ij} en un temps polynômial $0(n^3)$.

algorithme 2.5.1 PC-0

```
Procédure PC-0(X,D,C)
début
  pour k ← 0 à n faire
    pour i ← 0 à n faire
      pour j ← 0 à n faire
        C_ij ← C_ij ⊕ (C_ik ⊗ C_kj)
        si D_ij = ∅ alors arrêt          !inconsistance !
      finpour
    finpour
  finpour
fin
```

2.5.4 Propagation pour les TCSP

L'algorithme (2.5.1) présenté précédemment est sain pour les problèmes temporels généraux (les dates supprimées sont bien inconsistantes), mais il n'est pas complet (les domaines des variables ne sont pas réduits à leur minimum). Afin d'obtenir un TCSP sur lequel plus aucun ajustement des contraintes n'est possible, plusieurs passes sont nécessaires. L'algorithme *PC-1* (algorithme 2.5.2) permet de trouver le point fixe (toutes les déductions ont été faites, les domaines des variables ne peuvent plus être réduits).

On retiendra de l'algorithme *PC-1* deux inconvénients :

- il n'est toujours pas complet,
- les domaines des contraintes sont fragmentés (davantages d'intervalles).

Le problème de la fragmentation est illustré avec l'exemple de la figure 2.4, cette figure répresente un CSP temporel avant l'application de *PC-1* et la figure 2.5 après.

Dans le but de réduire cette fragmentation, dans [Schwalb et al.97] deux méthodes sont proposées :

algorithme 2.5.2 PC-1

Procédure PC-1(X,D,C)
début
  $fin \leftarrow faux$
  Tant que fin = faux **faire**
    $fin \leftarrow vrai$
    pour $k \leftarrow 0$ à n **faire**
      pour $i \leftarrow 0$ à n **faire**
        pour $j \leftarrow 0$ à n **faire**
          si $C_{ij} \neq C_{ij} \oplus (C_{ik} \otimes C_{kj})$ **alors**
            $fin \leftarrow faux$
            $C_{ij} \leftarrow C_{ij} \oplus (C_{ik} \otimes C_{kj})$!propagation !
          si $D_{ij} = \emptyset$ **alors arrêt** !inconsistance !
        finpour
      finpour
    finpour
  fin Tant que
fin

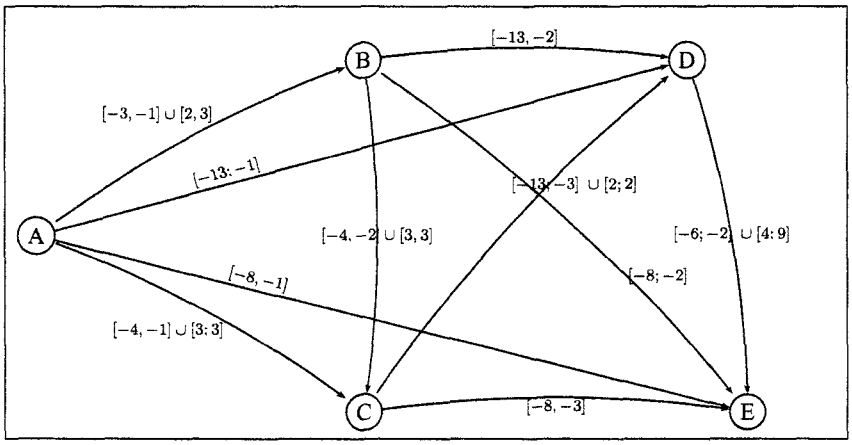

FIG. 2.4 – Exemple de graphe suite à la propagation locale de contraintes

- La première méthode s'appuie sur une relaxation des contraintes binaires à des contraintes formées d'un seul intervalle recouvrant l'ensemble des valeurs possibles. Cela consiste donc à transformer le TCSP en STP. Ainsi, chaque contrainte $C_{ij} : x_j - x_i \in \{ I_{ij}^1, ..., I_{ij}^{n_{ij}} \}$ devient $C'_{ij} : x_j - x_i \in [\min_{k=1..n_{ij}} (I_{ij}^k), \max_{k=1..n_{ij}} (I_{ij}^k)]$, où I_{ij}^k représente l'intervalle de valeurs admissibles.
  Cet algorithme est appelé *ULT* pour *Upper-Lower Tightening*. Cette relaxation permet de se ramener à un STP sur lequel *PC-0* détermine le réseau minimal.

  Exemple ULT

L'exemple ci-dessous présente le résultat de la transformation des contraintes par la méthode ULT :

$$C_{ij} = \{[0,22],[23,33],[34,50]\} \Rightarrow C'_{ij} = \{[0,50]\},$$

$$C_{ik} = \{[1,2],[11,12],[21,22]\} \Rightarrow C'_{ik} = \{[1,22]\},$$

$$C_{kj} = \{[0,1],[16,17],[23,24]\} \Rightarrow C'_{kj} = \{[0,24]\}.$$

- La seconde méthode appelée *LPC* (*Loose Path Consistency*) consiste à définir une nouvelle opération d'intersection entre 2 contraintes binaires $C^1_{ij} : x_j - x_i \in \{ I^1_1 \cup \ldots \cup I^1_{n_1}\}$ et $C^2_{ij} : x_j - x_i \in \{ I^2_1 \cup \ldots \cup I^2_{n_2}\}$. Cette contrainte $C^3_{ij} = C^1_{ij} \triangleleft C^2_{ij}$ est définie par l'ensemble des intervalles : $x_j - x_i \in \{ I^3_1 \cup \ldots \cup I^3_{n_3}\}, n_3 \leq n_1$, telle que $\forall k = 1,..,n_3, I^3_k = [lb^3_k, ub^3_k]$ où lb^3_k et ub^3_k sont les bornes minimales et maximales de l'intersection : $I^1_k \oplus (I^2_1 \cup \ldots \cup I^2_{n_2})$. Il est à noter que cette opération n'est pas commutative.

Exemple sur Loose Path COnsistency (LPC)
Soit les trois contraintes binaires de l'exemple précédent :
$C_{ij} = [0,22],[23,33],[34,50]$,
$C_{ik} = [1,2],[11,12],[21,22]$,
$C_{kj} = [0,1],[16,17],[23,24]$.
L'application de LPC sur la contrainte C_{ij} via la variable k donnera : $C_{ij} \leftarrow C_{ij} \triangleleft (C_{ik} \oplus C_{kj})$, soit $C_{ij} = [1,22],[23,29],[34,46]$

L'algorithme basé sur ce principe correspond à l'algorithme *PC-1* (algorithme 2.5.2) en remplaçant $\oplus$ par $\triangleleft$. Cet algorithme n'augmente pas le nombre d'intervalles des contraintes (union d'intervalles), comme le montre le graphe de la figure 2.6.

2.5.5 Résumé

Différentes définitions et algorithmes de propagation pour les TCSP ont été décrites dans cette partie. L'algorithme *PC-1* est très efficace pour réduire les domaines, mais il augmente la fragmentation. *ULT* est rapide d'exécution mais le filtrage est faible. *LPC* est un bon compromis entre temps de résolution et filtrage opéré.

La partie suivante présente une méthode de recherche arborescente pour les CSP.

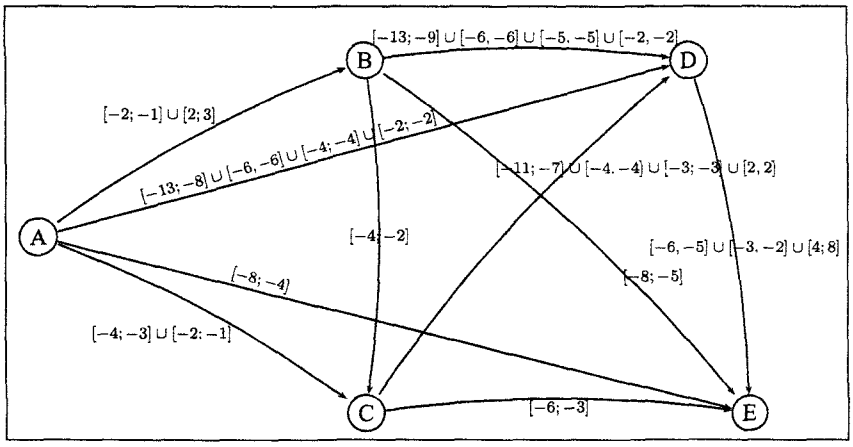

FIG. 2.5 – Graphe 2.4 après application de PC-1

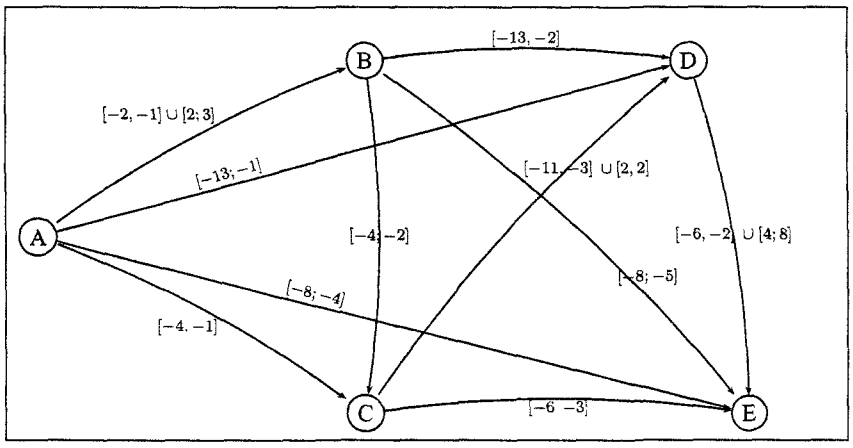

FIG. 2.6 – Graphe de la figure 2.4 après application de LPC

2.6 Recherche à divergence limitée

Les méthodes de recherche arborescente basées sur le concept de divergence sont utilisées depuis quelques années. Une divergence correspond au choix du nœud successeur qui n'est pas le premier préconisé par l'heuristique de choix de valeurs. Ces méthodes partent de l'idée que l'heuristique de choix de valeurs est très efficace et ne commet donc que très peu d'erreurs (ainsi le nœud ayant la meilleure évaluation devrait être le bon). Les stratégies de recherches basées sur les divergences considèrent, dans leur globalité, les divergences pouvant être effectuées pour parvenir à une solution. Les branches de l'arbre de recherche sont ainsi explorées dans un ordre plus pertinent qu'une recherche en profondeur d'abord. Pour un problème à variable binaire, le comptage des divergences est trivial : une divergence correspond à ne pas suivre le choix du nœud proposé par l'heuristique. Par contre pour un problème à variables entières, 2 possibilités de comptage existent :

1. Soit une divergence correspond à ne pas choisir la première valeur proposée par l'heuristique

de choix de valeur en chaque nœud : comptage 1. Il ne peut y avoir qu'une divergence par nœud.

La figure 2.7 représente le nombre de divergences accumulées en chaque nœud d'un arbre ternaire complet tel que proposé dans [Prcovic02]. Le choix heuristique par défaut est le nœud fils le plus à gauche, ainsi à chaque fois qu'un nœud à droite est exploré le nombre de divergences augmente de 1.

2. Soit à chaque choix de valeur, une divergence est ajoutée si le choix heuristique est remis en cause : comptage 2. La figure 2.8 représente le nombre de divergences accumulées en chaque nœud d'un arbre ternaire complet suivant cette dernière méthode de comptage.

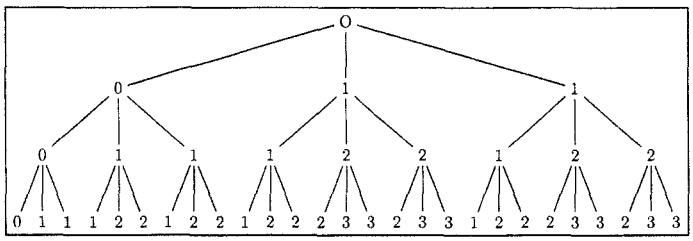

FIG. 2.7 – Nombre de divergences sur chaque nœud d'un arbre ternaire complet (comptage 1)

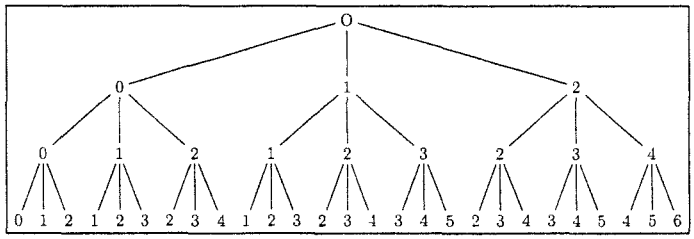

FIG. 2.8 – Nombre de divergences sur chaque nœud d'un arbre ternaire complet (comptage 2)

2.6.1 Limited Discrepancy Search

La recherche à divergence limitée, proposée par [Harvey et al.95] examine en premier lieu les branches de l'arbre offrant le moins de divergences par rapport à l'heuristique de choix de valeur. L'algorithme complet consiste à faire une recherche sans dépasser un nombre de divergences fixé, puis de recommencer en incrémentant le nombre de divergences autorisé. Un inconvénient de LDS est la redondance dans la génération des nœuds, ainsi Korf a proposé une amélioration à LDS, appelée "Improved LDS" (ILDS) [Korf96] et présentée par l'algorithme 2.6.1 où *discrepancy* fait référence au nombre de divergences autorisé à chaque itération. Cet algorithme permet d'éviter de se diriger vers les feuilles d'une profondeur donnée dont la divergence est supérieure à celle de l'itération courante. La figure 2.9 présente l'évolution de la recherche de l'algorithme ILDS sur un arbre binaire de profondeur 4.

algorithme 2.6.1 procédure ILDS

```
fonction ILDS(nœud)
  pour x ← 0 à profondeur_maximale faire        !profondeur maximale de l'arbre de recherche !
      résultat ← ILDS-itération(nœud, x, profondeur_maximale)
      Si résultat ! = pas de solution alors
        retourne résultat
finsi
  finpour
  retourne ∅
fin ILDS

fonction ILDS-itération(nœud, discrepancy, profondeur)
début
  Si nœud est une solution alors
      retourne nœud
  finsi
  s ← successeur(nœud)
  Si s == ∅ alors
      retourne pas de solution
  finsi
  Si profondeur > discrepancy alors
      retourne ILDS(s, discrepancy, profondeur-1)
  finsi
  Si discrepancy > 0 alors
      retourne ILDS(fils-droit(s), discrepancy-1, profondeur-1)
  finsi
fin ILDS-itération
```

2.6.2 Depth-bounded Discrepancy Search

La méthode Depth-bounded Discrepancy Search (DDS), proposée par Walsh en 1997 [Walsh97], est inspirée de LDS. Elle considère en plus le fait que l'heuristique de choix de valeurs risque de faire plus d'erreurs au début de l'arbre qu'à la fin et permet donc d'explorer plus de nœuds au début de l'arbre de recherche. L'heuristique de choix de valeurs a, en effet, une moindre connaissance de la qualité de la solution en cours d'élaboration au début de l'arbre qu'à la fin. Et ceci est d'autant plus vrai lorsque cette heuristique de choix de valeurs est dynamique, c'est à dire que les choix de valeurs à affecter sont calculés pendant la recherche. La méthode DDS fonctionne comme suit : à la première itération, DDS explore les feuilles ayant 0 divergence par rapport au choix heuristique. À la $i+1^{ième}$ itération, DDS explore les branches dont les divergences apparaissent à la profondeur i de l'arbre de recherche.

La figure 2.10 présente l'évolution de la recherche de l'algorithme DDS sur un arbre binaire de profondeur 4, $depth = 1$ et $discrepancy > 4$.

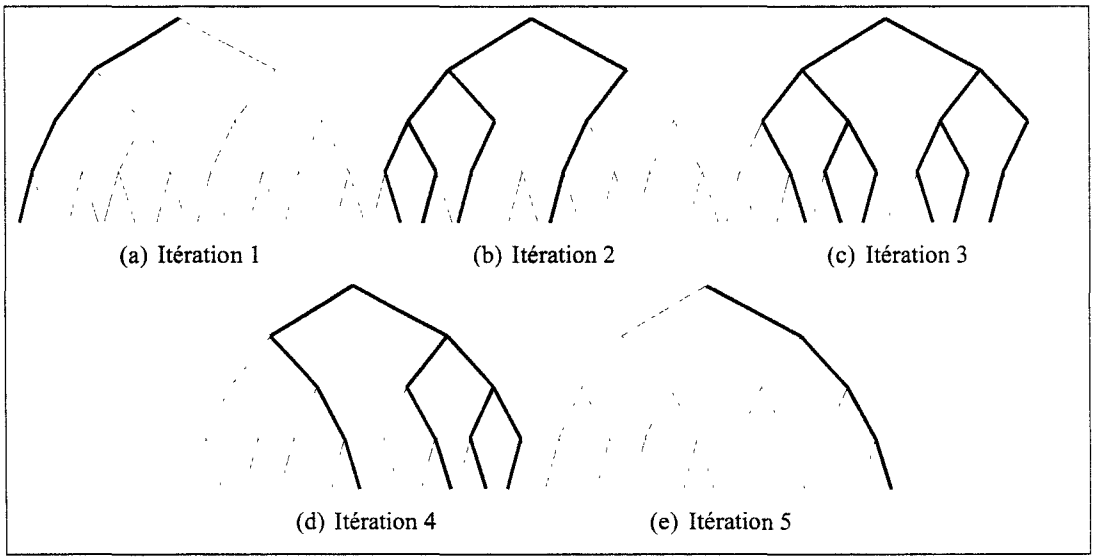

FIG. 2.9 – Itérations de ILDS

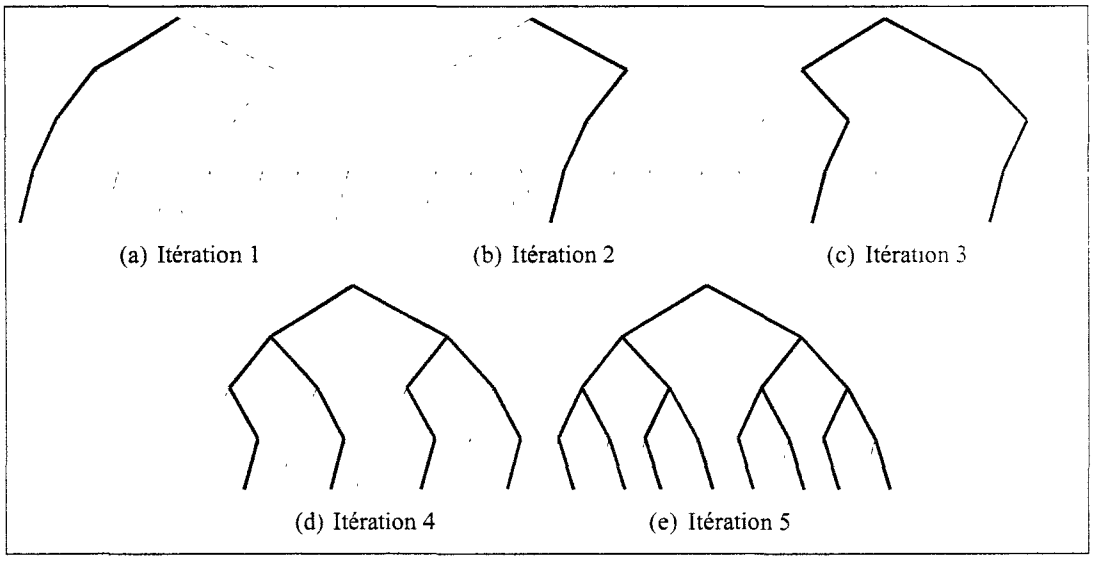

FIG. 2.10 – Itérations de DDS

2.6.3 Résumé

Cette section a présenté deux méthodes de recherches à divergence limitée. Il est également possible d'utiliser ces méthodes pour un algorithme de recherche incomplet en fixant une valeur limite au nombre de divergences tolérées : *discrepancy*. Ainsi seuls les nœuds les plus prometteurs sont parcourus. La section suivante présente les principaux algorithmes de résolution approchées de la programmation par contraintes.

2.7 Résolution approchée

Certains problèmes sont trop complexes pour être résolus dans le temps imparti ou dans l'espace mémoire machine disponible. En effet la résolution de problèmes implique l'énumération complète de toutes les solutions possibles. D'autres problèmes sont si complexes que leur résolution exacte demanderait plusieurs années. Dans ce cas, des techniques permettant d'obtenir une solution tenant compte des ressources de calcul limitées doivent être fournies. Ce type de technique ne permet pas de garantir l'optimalité de la solution, mais elles doivent fournir une solution de bonne qualité. En général le temps fait défaut, aussi ces techniques permettent de fixer le temps de recherche. La qualité de la solution dépendra fortement du temps accordé à sa résolution. Ces techniques sont appelées résolutions approchées ou heuristiques[2]. Reeves [Reeves95] propose la définition suivante d'une heuristique :

Définition 17 (Heuristique). *Une heuristique est une technique trouvant de bonnes solutions (proches de l'optimum) pour un coût de calcul raisonnable, sans pouvoir garantir la réalisabilité ou l'optimalité de la solution générée, ou même dans de nombreux cas, sans en préciser la distance à l'optimum.*

Les méthodes de résolution heuristique sont utiles pour des problèmes nécessitant une réponse en temps restreint ainsi que pour résoudre des problèmes difficiles (de par leur particularité ou la taille des instances). Ces méthodes peuvent également être utilisées pour aider la recherche exacte (Branch & Bound) en tant que phase d'initialisation.

En amont de ces heuristiques sont apparues des méthodes dites métaheuristiques. Plusieurs définitions sont proposées pour décrire leurs spécificités par rapport aux heuristiques. Osman et Laporte [Osman et al.96] proposent une définition et la comparent à un processus pilotant une heuristique, voire plusieurs, afin de l'amener plus efficacement vers la solution optimale :

Définition 18 (Métaheuristique selon Osman et Laporte). *Une métaheuristique est un processus itératif de génération guidant une heuristique subordonnée en combinant intelligemment différents concepts. Ces concepts explorent et exploitent l'espace de recherche en utilisant des stratégies pour structurer l'information de manière à trouver efficacement des solutions proches de l'optimum.*

[2] du grec heuriskein = trouver

La définition proposée par Pirlot [Pirlot02] présente les métaheuristiques en tant que stratégie de recherche :

Définition 19 (Métaheuristique selon Pirlot). *Les métaheuristiques ne sont pas à proprement parler des heuristiques, mais des schémas généraux, des "moules" à heuristiques ; le schéma général doit être adapté à chaque type particulier de problème.*

Ainsi les heuristiques ciblées sur un problème donné sont à différencier des métaheuristiques plus générales et adaptables aux problèmes. Bien évidemment l'efficacité des métaheuristiques dépendra également de l'adaptation faite au problème donné.

Il existe de nombreuses métaheuristiques dans le domaine de la recherche opérationnelle (algorithmes génétiques, recuit simulé, recherche tabou, GRASP, ...). Dans le domaine de la programmation par contraintes les métaheuristiques les plus utilisées sont à base de recherche locale, voisinage large ou recherche à arborescence tronquée (de type Limited Discrepancy Search) comme présenté en section 2.6.

2.7.1 Recherche locale et programmation par contraintes

Ces dix dernières années ont vu l'émergence d'un grand nombre de travaux combinant la programmation par contraintes et la recherche locale, on peut citer les principaux travaux : [Pesant et al.96], [Focacci et al.01a], [Pesant et al.99], [Rousseau et al.99] ou encore [Bohlin02]. Nous nous intéresserons ici uniquement à l'hybridation recherche locale / programmation par contraintes.

La recherche locale est utilisée, en recherche opérationnelle, depuis plus de trente ans. Elle est appliquée à de nombreux problèmes difficiles avec succès. Cette métaheuristique permet de trouver de des solutions en partant d'une solution initiale, celle ci est ensuite améliorée par des mouvements locaux. Ces petites améliorations successives peuvent amener à un optimum local. Un optimum local correspond à une solution de qualité meilleure que les solutions peu différentes mais moins bonnes que la solution optimale (cf. figure 2.11). Des stratégies permettent de sortir des optima locaux, notamment les métaheuristiques de recherche tabou [Glover et al.93] et recuit simulé [Kirkpatrick et al.83].

La recherche locale nécessite un opérateur définissant un voisinage. Un opérateur de voisinage peut être vu comme une manière de modifier une solution dans le but d'en trouver une meilleure. Le voisinage correspond à l'ensemble des solutions pouvant être obtenues en appliquant cet opérateur à une solution donnée. La taille d'un voisinage est définie par le nombre de solutions qu'il contient, un déplacement consistant à effectuer la transition de la solution courante à une solution du voisinage. Plus le voisinage sera important et plus il y aura de chances d'inclure de meilleures solutions mais plus le temps d'exploration sera grand.

L'hybridation de la recherche locale et de la programmation par contraintes généralement proposée utilise généralement l'algorithme de résolution par contraintes pour examiner le voisinage d'une solution et déterminer s'il en contient de meilleures.

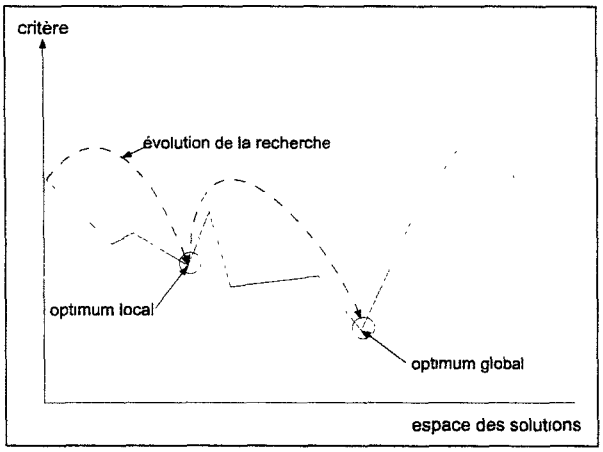

FIG. 2.11 – Courbe de l'objectif sur l'espace des solutions

2.7.2 Métaheuristique Large Neighborhood Search

La métaheuristique Large Neighborhood Search (recherche à large voisinage), notée LNS, a été proposée par Shaw [Shaw98]. Il s'agit d'une méthode par relaxation et ré-optimisation successive. À partir d'une solution initiale, cette méthode "libère" certaines variables (définies par l'opérateur) à chaque étape, puis une phase d'optimisation est effectuée. L'intérêt de la programmation par contraintes sur cette métaheuristique réside dans la propagation des variables déjà fixées, cette propagation permet de réduire fortement l'espace de recherche.

Le nombre de relaxations et ré-optimisations est dépendant d'un critère d'arrêt. Ce critère peut être :

- un nombre d'itérations fixé *a priori*, correspondant au nombre de phases de relaxation et ré-optimisation,
- un temps de résolution maximum,
- la qualité de la solution obtenue,
- une combinaison de ces critères.

Le sous-problème posé à chaque itération de l'algorithme LNS est alors optimisé par un algorithme de programmation par contraintes (propagation de contraintes, méthode arborescente, heuristique) en fixant des valeurs aux variables relâchées. Les étapes essentielles de cet algorithme sont au nombre de trois :

1. trouver une solution initiale pour la recherche locale,
2. définir un opérateur modifiant la solution initiale (choix des variables à relâcher),
3. explorer le voisinage à l'aide d'un algorithme de recherche de programmation par contraintes (choix de la variable "libre" à instancier, valeur à affecter, propagation).

La métaheuristique LNS, comme la majorité des méthodes de résolution approchées, requiert des paramètres pour la résolution, ces paramètres sont décrits dans les sections suivantes.

Solution initiale

Sa qualité va influer directement sur la qualité de la solution finale et/ou sur l'effort que devra fournir l'algorithme de recherche pour trouver une meilleure solution. Cette solution devrait répondre à deux principaux critères :

1. Ne pas être un optimum local non global par rapport au schéma d'exploration de LNS afin de ne pas bloquer la recherche.
2. Être de bonne qualité afin d'éviter à l'algorithme de recherche de passer trop de temps pour le cheminement qui va d'une solution de mauvaise qualité à une solution de bonne qualité.

La solution initiale peut être obtenue de différentes manières, la seule contrainte étant qu'elle doit être réalisable ; en voici trois possibles :

- une solution réalisable générée aléatoirement : elle présente l'avantage de ne pas avoir de particularité *a priori*, mais sa qualité risque d'être très mauvaise.
- une solution obtenue par une heuristique/métaheuristique : l'avantage est sa qualité, malheureusement elle risque d'être un optimum local. De plus le temps d'obtention de cette solution implique un temps moins important accordé aux autres étapes de LNS.
- une solution obtenue grâce à la connaissance des spécificités du problème : l'expérience du problème permet quelques fois d'obtenir une solution rapidement et de bonne qualité. Cette situation est malgré tout assez rare et ne peut pas se généraliser.

Type de voisinage

Le type de voisinage correspond au choix des variables à libérer. Ce choix est très important. En effet, si les variables libérées n'ont aucune contrainte commune alors leur affectation n'influera pas sur les autres variables libérées. De ce fait, la solution ne sera pas ou peu améliorée par le processus de recherche car les conflits entre variables ne seront pas changés. Par contre si les variables libérées sont fortement contraintes, le traitement des conflits augmentera les chances d'améliorer la solution obtenue précédemment.

1. Taille du voisinage : La taille du voisinage correspond au nombre de variables libérées par ré-optimisation. Plus cette taille est importante et plus la solution finale sera bonne étant donné qu'un plus grand espace de recherche aura été exploré. Mais plus le voisinage est important et plus le temps requis par chaque ré-optimisation est important. Un compromis doit donc être trouvé entre taille du voisinage et temps de résolution. Ce compromis est généralement obtenu grâce à la connaissance du problème.
2. Exploration du voisinage : Le but de LNS est d'explorer le voisinage de la manière la plus complète possible (recherche à voisinage large). Ainsi les méthodes de résolution utilisées sont en général :
   - une recherche exacte (cf. section précédente 2.4),

– une recherche à arborescence tronquée (cf. section suivante 2.6).

L'algorithme 2.7.1 résume les principales étapes de Large Neighborhood Search.

algorithme 2.7.1 Large Neighborhood Search

Procédure LNS
début
Créer une solution initiale
tant que *Critère d'arrêt = faux* **faire**
Sélectionner un ensemble E de variables du problème.
Ré-optimisation sur les variables libres variables ($X \notin E$).
Si *la recherche a amélioré la solution* **alors**
Mettre à jour la solution.
finsi
fintant
fin

2.7.3 Variable Neighborhood Search [Mladenovic et al.97]

Variable Neighborhood Search (VNS), dont une présentation complète ainsi que ses dérivés a été faite par Mladenovic [Mladenovic et al.97], est une métaheuristique utilisant les méthodes de recherche locale. Elle permet de changer le voisinage considéré lorsqu'un optimum local (cf. schéma 2.11) est atteint mais également de choisir le type de voisinage à exploiter. Ce schéma est inspiré des observations suivantes :

– Un optimum local peut être propre à un voisinage considéré, l'utilisation d'autres types de voisinage peut permettre de sortir de cet optimum local.
– Un optimum global est un optimum local par rapport à toutes les structures de voisinage.
– Pour une majorité de problèmes les optimums locaux par rapport à un ou plusieurs voisinages sont relativement proches les uns des autres.

Comparativement aux autres métaheuristiques, les schémas généraux de VNS et de ses extensions ne requièrent que peu de paramètres. Une présentation des différentes extensions de VNS est proposée par la suite, Hansen et Mladenovic [Hansen et al.03] proposent une bibliographie complète de VNS et de ses déclinaisons.

Variable Neighborhood Descent Search (VND)

Cette méthode consiste, en partant d'une solution initiale x, à trouver la meilleure direction d'amélioration de la solution (descente) en considérant un voisinage $N_k(x)$, k_{max} représentant le nombre de types de voisinage. L'algorithme 2.7.2 présente les principales étapes de VNS.

algorithme 2.7.2 Variable Neighborhood Descent

Procédure VND
début
  $k \leftarrow 1$
  tant que $k < k_{max}$ **faire**
    Trouver le meilleur voisin x' du voisinage $N_k(x)$ considéré.
    Si *la solution n'est pas améliorée* **alors**
      $k \leftarrow k + 1$
    finsi
    Si *la solution est améliorée* **alors**
      $k \leftarrow 1$
    finsi
  fintant
fin

De par cette méthode, la solution finale doit être un optimum local pour chaque voisinage et ainsi elle sera meilleure (ou tout au moins aussi bonne) que si une seule structure de voisinage était testée.

Reduced Variable Neighborhood Search (RVN)

VND peut présenter des limites lorsque la taille des instances implique un temps de calcul trop important pour obtenir les voisins. Ainsi RVN se différencie de VND en explorant non pas l'ensemble du voisinage mais un voisin choisi aléatoirement. La recherche est donc réduite et peut être qualifiée de stochastique étant donné le caractère aléatoire du choix du voisin. Les autres étapes restent inchangées.

General Variable Neighborhood Search (GVNS)

Cette méthode combine la recherche stochastique (RVN) et déterministe (VND). Suite au choix aléatoire du voisin, une boucle VND est appliquée sur ce voisin avec des structures de voisinages différents ou identiques à la boucle RVN. Le critère d'arrêt peut être soit un temps maximum, un nombre d'itérations, ou un nombre maximum d'itérations entre deux améliorations. Cette méthode est celle présentant les meilleurs résultats en général comme indiqué dans [Hansen et al.00] par rapport aux autres méthodes VNS.

2.8 Conclusion

Ce chapitre a présenté les problèmes de satisfaction de contraintes et notamment les problèmes de satisfaction de contraintes temporelles. Différentes techniques de résolution de la programma-

tion par contraintes ont été décrits : filtrages par consistance, résolutions exactes et approchées. La programmation par contraintes est un domaine de l'optimisation permettant de résoudre un grand nombre de problèmes. Les techniques de filtrages proposées permettent de réduire fortement l'arbre de recherche. Les techniques de résolution approchée permettent de trouver une solution (si l'on se place dans un objectif d'optimisation) de bonne qualité en un temps satisfaisant.

Dans la suite, nous nous intéresserons à la modélisation proposée pour le problème d'évaluation de capacité d'infrastructures ferroviaires défini en section 1. Ce problème est modélisé en un problème de satisfaction de contraintes comportant des contraintes temporelles. Ensuite les techniques utilisées pour la résolution de ce problème sont détaillées. Elles utilisent les techniques de programmation par contraintes étudiées dans ce chapitre.

Chapitre 3

Une modélisation du problème de capacité ferroviaires

Ce chapitre développe les différents éléments de la modélisation adoptée afin de couvrir les spécifications du problème décrit au chapitre 1. Les différentes méthodes traitant ce problème, dont celles présentées au chapitre 1, ne satisfont pas totalement les objectifs de l'évaluation de la capacité fixés pour ce travail. En effet, mis à part le modèle du projet RECIFE, la modélisation des différents projets étudiés n'est pas assez fine pour la résolution du problème de saturation au niveau d'un nœud. Les hypothèses trop réductrices qui sont prises conduisent à des solutions qui, soit ne sont pas totalement réalisables, soit sont loin de l'optimum. Le projet RECIFE, dont ce travail prend la suite, propose une modélisation fine du problème, par une modélisation de type Set Packing (problème à contraintes de disjonction et variables booléennes) avec un pas de discrétisation du temps relativement petit. Les instances étudiées sont résolues de manière correcte au niveau d'un nœud pour le problème de saturation. La question nous ayant été posée pour la suite de ce projet est la suivante : Est il possible de trouver une autre modélisation au moins aussi fine permettant de donner des résultats de meilleure qualité ?

L'approche proposée s'inspire des travaux de Fukumori et Sano [Fukumori et al.87] et plus récemment, du travail de Hachemane [Hachemane97]. Elle est basée sur la gestion des conflits entre couples de trains.

Avant de présenter la modélisation, un rappel sur la gestion des conflits est effectuée, puis l'ensemble des variables du problème sont introduites. Ensuite, les contraintes permettant de traiter le problème de saturation en tenant compte des règles de sécurité sont présentées. Un certain nombre d'améliorations au modèle sont proposées. Enfin une présentation des différents modèles utilisables, les formats de données et les spécificités du problème de saturation et du problème de faisabilité sont décrits.

3.1 Gestion des conflits de circulations

La définition du problème de saturation utilisée est celle de Delorme [Delorme03], présentée en 3, section 1.3.2.

Ce type de problème amène donc à l'élaboration d'une grille horaire saturée, c'est à dire à établir les heures de passage des trains sur les voies de la manière la plus dense possible. La circulation des trains sur ces voies doit être effectuée en toute sécurité, ainsi tout conflit doit être géré. L'arbitrage des conflits est effectué grâce à l'ordonnancement des trains sur l'infrastructure. C'est à partir de l'arbitrage des conflits que les passages des trains et donc les écarts minimaux entre trains sont déterminés. Ainsi il constitue la base de l'établissement de grille horaire, saturée ou non. La suite de cette section présente une définition précise des conflits et la manière dont leur arbitrage peut être modélisé.

3.1.1 Notations

Un certain nombre de notations est nécessaire afin de donner une description plus formelle de la gestion d'un conflit :

- T : l'ensemble des trains,
- $N = card(T)$: le nombre de trains,
- Z : l'ensemble des zones de l'infrastructure. Une zone est la plus petite entité permettant la détection de la présence des trains sur la voie. Le dispositif le plus utilisé pour obtenir cette information est un dispositif électrique appelé "circuit de voie" (cf. figure 1.4, section 1.1)
- It : l'ensemble des itinéraires de l'infrastructure,
- G : l'ensemble des types de trains possibles (TGV, trains corails, trains de marchandise),
- R : l'ensemble des itinéraires associé au type de trains ($R = G \times It$),
- r_i, $i \in T$, $dom(r_i) \in R$: variable représentant à la fois l'itinéraire emprunté et le type du train i. Ainsi une même variable indique l'itinéraire qu'emprunte un train, mais également ses caractéristiques cinématiques (longueur, vitesse). Ce choix de modélisation permet d'avoir une relation univoque avec la durée d'occupation des différentes ressources de l'infrastructure. Par la suite nous nommerons cette variable r_i "variable parcours".
- $\underline{t_{ij}}$ et $\overline{t_{ij}}$: variables représentant respectivement les bornes supérieures et inférieures des intervalles entre les dates de départ des trains i et j dans l'infrastructure.
- $\underline{I}$ et $\overline{I}$: les ensembles des valeurs possibles respectivement pour toutes les variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$.
- st_i : variable représentant la date d'entrée du train i dans l'infrastructure, le pas de discrétisations est à la seconde, ainsi $st_i \in \mathbb{N}$.

3.1.2 Définition d'un conflit

La gestion du conflit de circulation est centrale dans la gestion du trafic ferroviaire. Un conflit traduit une incompatibilité entre les choix de parcours et d'heures d'entrée des trains dans une infrastructure. On appelle conflit de circulation (cf. figure 3.1) la séquence des zones communes à deux parcours. Soit p_1 et p_2 deux parcours, z_a et z_b la première et la dernière zone du conflit dans le sens de circulation du parcours p_1, la définition d'un conflit de circulation est alors :

Définition 20 (Conflit de circulation). *[Rodriguez02] Un conflit de circulation est un quadruplet* $(p_1, p_2, z_a, z_b) \in R \times R \times Z \times Z$ *tel que l'ensemble des zones entre* z_a *et* z_b *du parcours* p_1 *est égal à l'ensemble des zones entre* z_a *et* z_b *du parcours* p_2.

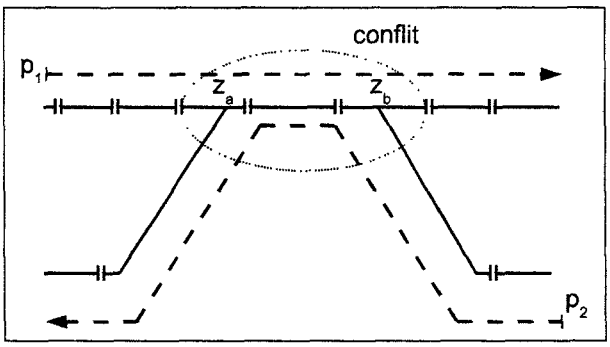

FIG. 3.1 – Un exemple de conflit

De manière plus générale, dans le contexte de l'ordonnancement, un conflit se produit lorsque des activités requièrent une ressource au-delà de sa capacité. Deux approches de modélisation sont fréquemment utilisées : soit les conflits sont implicites à travers des contraintes de capacité des ressources, soit les conflits sont explicitement décrits entre les activités.

Dans le premier cas l'identification d'un conflit se traduit par le fait que les contraintes entre les variables de capacité, d'affectation et d'utilisation des ressources ne peuvent être satisfaites. Le conflit conduit à explorer une autre branche de l'arbre de recherche.

Dans le second cas, chaque conflit est présent dans le modèle du problème et prend la forme d'une contrainte disjonctive. Cette contrainte porte sur les variables d'affectation et d'utilisation d'une ressource. Le modèle ne comporte plus de variables sur la capacité des ressources.

Sur cette étude, nous avons opté pour une modélisation explicite des conflits.

3.1.3 Les différents types de conflits

Comme présenté en chapitre 1, l'infrastructure ferroviaire peut présenter des topologies très variées. Certaines parties de l'infrastructure permettent la circulation des trains, soit dans un seul sens donné, soit dans les deux sens. Cette possibilité amène à distinguer deux catégories de conflits :

- le *conflit d'espacement* pour lequel les trains circulent dans le même sens sur la séquence de zones communes ; l'ordre de parcours des zones est identique pour les deux trains (cf. figure 3.2),
- le *conflit de contre-sens* dans lequel les trains circulent dans des sens opposés et donc où l'ordre de parcours des zones communes est inversé, comme présenté en figure 3.3.

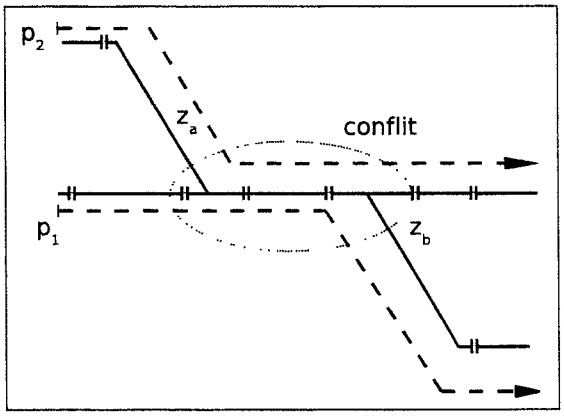

FIG. 3.2 – Un conflit d'espacement

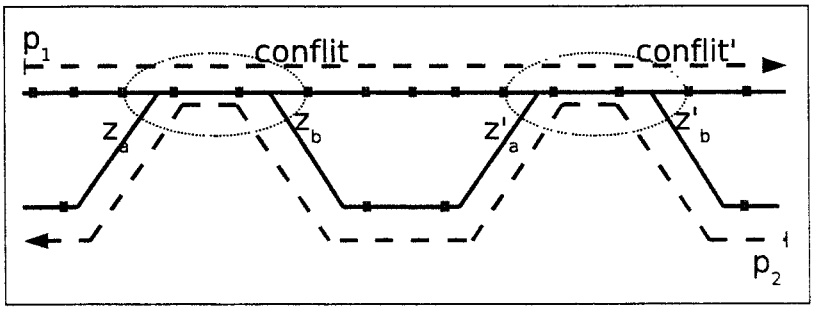

FIG. 3.3 – Deux conflits de contre-sens

3.1.4 Conflits

Lorsque deux trains convoitent au même instant une zone de l'infrastructure, un conflit a lieu. Il convient alors de choisir l'ordre de passage de ces trains et de considérer les différents cas de figure.

Arbitrage des conflits

L'ordonnancement des trains qui empruntent des itinéraires (p_1 et p_2) comportant une zone commune (entre z_a et z_b) doit permettre d'éviter toutes collisions. Il convient alors d'arbitrer l'ordre de passage de ces trains, l'arbitrage d'un conflit est défini comme suit :

Définition 21. *Un arbitrage d'un conflit de circulation* (p_1, p_2, z_a, z_b) *est le choix de l'ordre de passage d'un couple de trains empruntant les parcours* p_1, p_2 *sur la séquence de zones entre* z_a *et* z_b.

Comme la description ci-dessus le préconise, l'ordonnancement des trains sur l'infrastructure est géré par couple de trains.

Dans le cadre d'études de planification, les trains ne peuvent pas subir de ralentissements, par conséquent, la date d'entrée des trains dans l'infrastructure sera la seule variable de décision pour l'arbitrage des conflits.

Prenons le cas d'un conflit de contre sens (les deux trains se font de face comme sur l'exemple de la figure 3.4), avec le train 1 prenant l'itinéraire p_1 et le train 2 l'itinéraire p_2. Ces deux trains ont une portion commune. Il faut donc ordonner les trains afin que :

- Si le train 1 passe en premier sur la zone commune, alors la sortie du train 1 de la zone commune précède l'entrée du train 2 dans la zone commune.
- Si le train 2 passe en premier sur la zone commune, alors la sortie du train 2 de la zone commune précède l'entrée du train 1 dans la zone commune.

Les temps de parcours des zones étant supposés identiques pour un type de trains et un parcours définis (parcours en voie libre), le temps de trajet des zones est donc constant et connu à l'avance. Pour la modélisation utilisée, les conflits sont gérés par l'intermédiaire d'intervalle de temps. Pour un conflit entre ces deux trains, deux alternatives sont alors possibles :

- le train 1 doit entrer dans l'infrastructure au minimum $\underline{t_{12}}$ secondes avant le train 2. Ce temps de décalage tient compte du temps de parcours de la zone commune par le train 1, mais également du temps de parcours entre le début de l'infrastructure et le début de la zone commune (soit T_{p1} et T_{p2} sur la figure). Il est à noter que ce temps peut être négatif, cela signifie que le train 1 doit entrer au plus tard $|\underline{t_{12}}|$ secondes après l'entrée du train 2. Ainsi, $\underline{t_{12}}$ correspond au décalage d'entrée du train 1 par rapport à l'entrée du train 2 dans l'infrastructure, si le train 1 passe avant.
- le train 2 doit entrer dans l'infrastructure au moins $\overline{t_{12}}$ secondes après le train 1. Ce temps de décalage tient compte du temps de parcours de la zone commune par le train 2, mais également du temps de parcours entre le début de l'infrastructure et le début de la zone commune (soit T_{p1} et T_{p2} sur la figure). De même que pour $\underline{t_{12}}$, ce temps peut être négatif, cela signifie que le train 2 doit entrer dans l'infrastructure au plus tôt $|\overline{t_{12}}|$ secondes avant le train 1. $\overline{t_{12}}$ correspond au décalage d'entrée du train 2 par rapport à l'entrée du train 1 dans l'infrastructure, si le train 2 passe avant.

L'intervalle $[\underline{t_{12}}, \overline{t_{12}}]$ tient compte des temps d'occupation de la zone commune par les trains 1 et 2, du temps de trajet pour atteindre la zone commune ainsi que d'un temps de sécurité. Il s'agit d'un intervalle d'incompatibilité entre la différence des dates d'entrées des trains dans l'infrastructure.

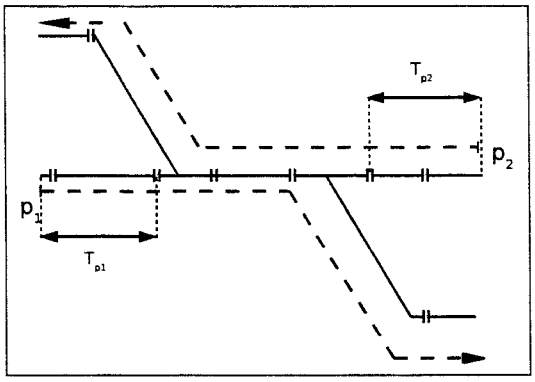

FIG. 3.4 – Exemple de conflit

Principes d'espacement et d'enclenchement

L'infrastructure ferroviaire présente différents cas de figure amenant à des conflits. Afin de tenir compte de ces différents cas, les dispositifs de sécurité ferroviaire s'appuient sur deux principes, chacun étant associé à une catégorie de conflit :

- Le principe d'espacement, il doit permettre de garder en toutes circonstances les trains circulant dans le même sens (cf. cas de la figure 3.2) à des distances supérieures à leur distance d'arrêt.
- Le principe d'enclenchement (cf. figure 3.3), il vise à imposer des conditions aux manœuvres des appareils de voie (telle une aiguille) et aux signaux de protections.

Comme nous venons de le voir, la formation des itinéraires, des enclenchements et des distances de freinages impose une séquence d'événements aux circulations des trains.

Si cette séquence d'événements ne peut être respectée lorsque les trains circulent avec la vitesse maximale autorisée, alors des retards par rapport aux horaires théoriques apparaissent.

Dans le cas d'un conflit d'espacement, considérons deux trains 1 et 2 empruntant des parcours p_1 et p_2 comme sur l'exemple de la figure 3.2, et ayant une zone commune dans le même sens de circulation, alors :

- $\underline{t_{12}}$ représente le temps d'espacement pour les dates d'entrées des trains dans l'infrastructure, ce temps tient notamment compte de la distance (et donc du temps) d'arrêt du train 2 avant la zone commune,
- respectivement $\overline{t_{12}}$ représente le temps d'espacement pour les dates d'entrées des trains dans l'infrastructure des trains, ce temps tient notamment compte de la distance (et donc du temps) d'arrêt du train 1 avant la zone commune,

Dans le cas d'un conflit de contre-sens, considérons deux trains 1 et 2 empruntant des parcours p_1 et p_2 comme sur l'exemple de la figure 3.3, et ayant une zone commune, alors :

- $\underline{t_{12}}$ tient compte du temps d'occupation de la zone commune, du temps d'arrivée du train sur la zone commune, du temps de dégagement de la zone par le train 1, ainsi que de la distance

(et donc du temps) d'arrêt du train 2 avant la zone commune,
- respectivement $\overline{t_{12}}$ représente le temps d'espacement nécessaire entre les trains, ce temps tient notamment compte de la distance (et donc du temps) d'arrêt du train 1 avant la zone commune,

Conflits multiples

Comme le présente la figure 3.3, plusieurs conflits peuvent avoir lieu entre deux mêmes trains, ainsi il peut y avoir plusieurs intervalles d'incompatibilités entre les dates d'entrées des trains dans l'infrastructure, soit K^{r_i,r_j} l'ensemble des incompatibilités associées au couple de parcours (r_i, r_j), nous avons alors pour le couple de trains i, j les intervalles d'incompatibilités :

$$]\underline{t_{ij}^k},\ \overline{t_{ij}^k}[,\ k = 1,\ \ldots,\ card(K^{r_i,r_j}). \tag{3.1}$$

Les conflits sont ainsi triés du conflit le plus pénalisant au moins pénalisant, le plus pénalisant étant celui dont l'intervalle d'incompatibilité est le plus grand :

$$]\underline{t_{ij}^1},\ \overline{t_{ij}^1}[,]\underline{t_{ij}^2},\ \overline{t_{ij}^2}[\ldots avec(\overline{t_{ij}^k} - \underline{t_{ij}^k}) > (\overline{t_{ij}^{k+1}} - \underline{t_{ij}^{k+1}}), \forall k = 1,\ \ldots,\ card(K^{r_i,r_j} - 1). \tag{3.2}$$

De cette manière les conflits vont être caractérisés par des décalages temporels sur les dates d'entrées des trains dans l'infrastructure. Ces décalages résultent en $card(K^{r_i,r_j})$ intervalles d'incompatibilité entre couples de trains.

La détermination de ces intervalles d'incompatibilité est obtenue (dans notre cas) grâce à une phase de simulation sur l'infrastructure considérée. Les données d'entrée sont les temps de parcours sur les zones de l'infrastructure, ils sont issus du simulateur SISYFE [Fontaine et al.01] (SImulateur du SYstème FErroviaire) développé par la SNCF. À partir de ces temps de parcours, les intervalles d'incompatibilité entre couples d'itinéraires et type de trains (représenté par la variable r_i) sont obtenus.

3.1.5 Hypothèses de l'étude

L'évaluation de la capacité est effectuée en tenant compte d'hypothèses sur le passage des trains sur l'infrastructure. Deux hypothèses sont utilisées, la première porte sur les parcours des trains sur l'infrastructure, l'autre sur le temps de trajet de ces parcours.

Hypothèse 1. *Les trains peuvent emprunter des parcours pré-définis, correspondant aux parcours réels de l'infrastructure étudiée, ces parcours sont ceux de l'ensemble R défini en section 3.1.1.*

Hypothèse 2. *Le temps de trajet des trains sur les parcours est supposé connu et toujours identique pour un itinéraire et un type de trains donnés.*

3.2 Modèle de base

La formulation adoptée pour la modélisation du problème de saturation est celle d'un problème de satisfaction de contraintes (cf. chapitre 2). Une autre formulation CSP de ce problème a été précédemment proposée par Rodriguez dans [Delorme et al.01]. Cette modélisation était établie à l'origine pour résoudre des problèmes de régulation du trafic. Son adaptation au problème de saturation implique l'ajout d'une contrainte forçant les trains à parcourir l'infrastructure en voie libre. Les résultats des expérimentations publiées dans [Delorme et al.01] montrent que ce modèle CSP a des performances comparables à celles du modèle de Set Packing (SPP). Le modèle CSP est même capable de fournir des solutions de meilleure qualité que le modèle SPP pour certaines instances. Le modèle SPP et les algorithmes de résolution ayant pu être par la suite développés et améliorés, la poursuite d'une recherche sur l'autre approche de modélisation s'imposait afin de poursuivre plus avant la comparaison et tirer des leçons des avantages respectifs de chaque approche. C'est pourquoi, à partir de quelques bases du modèle CSP, une nouvelle modélisation spécifiquement dédiée au problème de saturation a été developpée et présentée une première fois dans [Degoutin et al.05c]. Le détail de cette modélisation fait l'objet de cette section.

3.2.1 Variables

Une solution du problème de saturation est caractérisée par les itinéraires et types de trains choisis pour les trains ainsi que par l'ordonnancement des trains sur les conflits. Les variables nécessaires à la représentation de ce problème sont au nombre de trois et peuvent être classées en deux catégories :

1. Variables d'affectation de ressource :
   - **les variables "parcours"** $\boxed{r_i}$, avec $dom(r_i) \subset R$, $i \in T$. Ces variables représentent l'itinéraire et le type du train i. R inclut donc l'ensemble des combinaisons d'itinéraires et de type de trains possibles pour le trains i.
2. Variables d'ordonnancement des trains :
   - **les variables "bornes d'incompatibilités"** $\boxed{\underline{t_{ij}^k}, \overline{t_{ij}^k}}$, avec $dom(\underline{t_{ij}^k}) \subset \underline{I}$ et $dom(\overline{t_{ij}^k}) \subset \overline{I}$, $i \in T, k = 1, \ldots, card(K^{r_i,r_j})$ Ces variables représentent les valeurs des bornes des $card(K^{r_i,r_j})$ intervalles d'incompatibilités entre le couple de trains (i,j). Ces variables permettent de déterminer les écarts minimaux entre les dates d'entrées des trains dans l'infrastructure afin d'éviter les conflits.
   - **les variables "dates d'entrées"** $\boxed{st_i}$, avec $i \in T$.
     Le pas de discrétisation choisi est la seconde. Il est à noter que ce pas n'est pas réaliste en phase opérationnelle, en effet il est impossible qu'un conducteur puisse faire entrer son train à la seconde près. Il s'agit d'une discrétisation permettant d'obtenir des dates d'entrées des trains qui se traduisent ensuite comme une valeur théorique de la capacité

de l'infrastructure étudiée. Cette valeur théorique de la capacité est ensuite utilisée comme un élément de réflexion par les décideurs.

Les variables permettant la modélisation de ce problème ayant été introduites, la section suivante présente les principales contraintes de ce problème.

3.2.2 Contraintes

Un premier type de contraintes provient des contraintes techniques issues des dispositifs de sécurité, elles protègent les circulations contre les principaux risques de collision ou déraillement. Elles sont modélisées dans notre modèle par l'intermédiaire de deux ensembles de contraintes. Le premier ensemble permet de déterminer les intervalles de compatibilités entre couples de trains, le second permet d'arbitrer les conflits entre couples de trains. L'espace des solutions défini par ces deux ensembles de contraintes présente des symétries, aussi un troisième ensemble de contraintes a été introduit pour les éviter.

Contraintes d'énumération

Les contraintes techniques de sécurité conduisent à l'incompatibilité entre certaines dates d'entrée des trains lorsque ces derniers suivent des parcours qui ont des zones communes. L'ensemble de ces dates d'incompatibilité forme des intervalles. Le premier ensemble de contraintes établit le lien entre les variables parcours (r_i) et les variables bornes d'incompatibilité ($\underline{t_{ij}^k}$, $\overline{t_{ij}^k}$).

Ces contraintes sont établies par l'énumération des tuples admissibles entre les variables parcours et bornes d'incompatibilité des trains i, j. Si l'on note $enum(,,,)$, la fonction d'énumération des 4-uplets admissibles, les contraintes sur les intervalles d'incompatibilité peuvent donc s'exprimer de la façon suivante :

$$enum(r_i,\ r_j,\ \underline{t_{ij}^k},\ \overline{t_{ij}^k}),\ \forall i,j\ \in\ T^2,\ k=1,...,N_k. \tag{3.3}$$

avec $N_k = \max_{i,j \in T^2} card(K^{r_i,r_j})$.

Il peut exister plusieurs intervalles d'incompatibilités entre les couples de trains ($card(K^{r_i,r_j})$), mais ce nombre n'est connu que lorsque les parcours r_i et r_j sont fixés (ou tout au moins leur domaine réduit). Ainsi pour chaque couple de trains, le même nombre de contraintes est posée, cela implique l'emploi de variables et contraintes fictives pour un certain nombre d'entre eux. L'efficacité de la résolution s'en trouve alors réduite à cause de la propagation et la gestion de variables inutiles. De plus, le nombre de conflits multiples sur les infrastructures considérées est très faible (de l'ordre de 5%). C'est pourquoi nous avons choisi de ne poser que la contrainte correspondant au premier conflit (selon le classement donné par l'équation 3.2). De ce fait, seules les contraintes pour $k = 1$ seront posées, les valeurs des variables $\underline{t_{ij}^l}$ et $\overline{t_{ij}^l}$ avec $l > 1$ seront déduites directement des affectations des parcours r_i et r_j pour la pose des contraintes d'incompatibilités. De ce fait, les variables $\underline{t_{ij}^l}$ et $\overline{t_{ij}^l}$ avec $l > 1$ n'ont plus lieu d'être.

Nous avons alors la modélisation de cette contrainte d'énumération :

$$enum(r_i,\ r_j,\ \underline{t^1_{ij}},\ \overline{t^1_{ij}}),\ \forall i,j\ \in\ T^2, \tag{3.4}$$

Dans la suite de l'énoncé, nous ne considérerons généralement qu'un seul intervalle d'incompatibilité par couple de trains, ainsi $\underline{t_{ij}}, \overline{t_{ij}}$ représenteront les variables $\underline{t^1_{ij}}, \overline{t^1_{ij}}$.

Exemple contraintes d'énumération
Prenons l'exemple de deux trains 1 et 2 pouvant emprunter les trois parcours a, b et c de l'infrastructure de la figure 3.5, ainsi $dom(r_i) = \{a, b, c\}, i \in \{1, 2\}$. La phase de simulation fournit les écarts entre les parcours des trains pour une circulation en toute sécurité. Chaque couple d'itinéraires est alors testé afin de connaître pour chaque combinaisons de parcours les intervalles d'incompatibilités. En d'autres termes, la phase de simulation permet d'obtenir les correspondances entre les variables parcours (r_1 et r_2) et les variables bornes d'incompatibilités ($\underline{t^1_{ij}}$, $\overline{t^1_{ij}}$).
6 conflits différents sont possibles en comptabilisant les conflits entre trains empruntant les mêmes itinéraires (ils doivent se suivre avec un écart minimum d'espacement). Plusieurs cas de conflits sont envisageables, deux sont plus précisément décrits ici :

- Dans le cas où le train 1 emprunte le parcours a et le train 2 le parcours b, nous avons $r_1 = a$ et $r_2 = b$. Grâce à la phase de simulation, nous savons que ces trains doivent avoir un décalage d'au moins $70s$ si le train 1 entre en premier et d'au moins $90s$ si le train 2 entre en premier. Ces valeurs seront prises en compte dans cette contrainte en ajoutant le tuple admissible : $(a,\ b,\ -70,\ 90)$.
- Dans le cas où les deux trains empruntent le même parcours, le décalage temporel sera identique quelque soit le premier train entrant. La phase de simulation a indiqué que si $r_1 = r_2 = a$ alors le décalage temporel doit être de $80s$, nous aurons alors à ajouter le tuple admissible : $\{a, a, -80, 80\}$.

De cette manière nous obtenons l'ensemble des tuples admissibles pour cette infrastructure donnée dans le tableau 3.2.
Lors de la phase de résolution, si $r_i = a$ et $r_j = b$, alors la variable $\underline{t_{ij}}$ aura son domaine réduit au singleton $\{-70\}$ et la variable $\overline{t_{ij}}$ aura son domaine réduit à $\{90\}$. Et réciproquement, si $\underline{t_{ij}} = 80$, alors r_i et r_j auront leur domaine réduit à $\{a, c\}$.

Contraintes d'incompatibilités

Le second ensemble de contraintes modélise les contraintes d'incompatibilité entre les dates d'entrée des trains dans une infrastructure. Les contraintes d'incompatibilité transcrivent les alternatives de décision qui permettent d'arbitrer les conflits de passages sur les zones communes aux parcours de deux trains.

parcours 1	parcours 2	$\underline{t_{ij}^1}$	$\overline{t_{ij}^1}$
a	a	-80	80
a	b	-70	90
a	c	-100	120
b	b	-60	60
b	c	-150	210
c	c	-80	85

TAB. 3.2 – Exemple de tuples parcours bornes d'incompatibilités

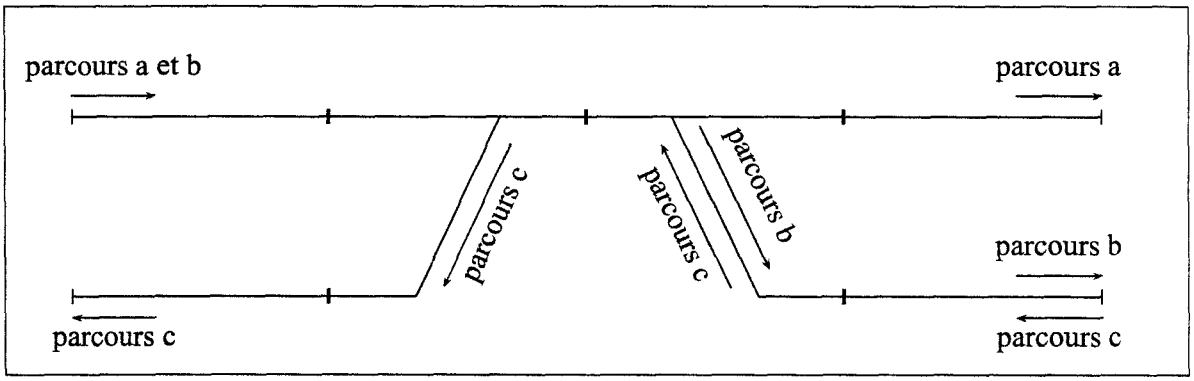

FIG. 3.5 – Exemple d'infrastrucure avec 3 parcours possibles

Dans ces contraintes, on relie les couples de variables bornes d'incompatibilité de la contrainte d'énumération de la section précédente aux variables dates d'entrée des trains.

L'arbitrage d'un conflit consiste à ordonner le passage d'un couple de train sur les zones communes. À chaque décision d'ordre correspond une contrainte potentielle entre les dates d'entrée des trains. Les deux décisions d'ordre entre les trains forment une contrainte disjonctive.

En ce qui concerne les conflits ayant des intervalles multiples, on remarque qu'ils donnent lieu à autant d'arbitrages qu'il y a d'intervalles. Par conséquent, on aura aussi pour ce type de conflits autant de contraintes disjonctives que d'intervalles.

En définitive, nous avons formulé l'ensemble des contraintes d'incompatibilité, qui sera nommé inc_1, par :

$$((st_i - st_j) \leq \underline{t_{ij}^k} \vee (st_i - st_j) \geq \overline{t_{ij}^k}), \forall i,j \in T^2,\ \forall i,j \in T^2,\ k = 1,....N_k. \qquad (3.5)$$

avec $N_k = \max_{i,j \in T^2} card(K^{r_i,r_j})$.

De même que pour la contrainte d'énumération précédente 3.4, il peut exister plusieurs intervalles d'incompatibilités entre les couples de trains ($card(K^{r_i,r_j})$ en tout), mais ce nombre n'est connu que lorsque les parcours r_i et r_j sont connus.

Pour les mêmes raisons que pour la contrainte d'énumération, nous avons choisi d'écrire la

contrainte d'incompatibilité sous la forme d'une contrainte classique et d'une méta-contrainte posée sous la condition que le couple de parcours possède plus d'un conflit potentiel. Nous avons alors la modélisation de cette contrainte d'incompatibilité :

$$((st_i - st_j) \leq \underline{t_{ij}^1} \vee (st_i - st_j) \geq \overline{t_{ij}^1}), \forall i, j \in T^2, \tag{3.6}$$

$$(r_i = a) \wedge (r_j = b) \rightarrow ((st_i - st_j) \leq \underline{t_{ab}^k} \vee (st_i - st_j) \geq \overline{t_{ab}^k}), \quad \forall i, j \in T^2, \forall k = 2, \ldots, card(K^{a,b}), \tag{3.7}$$

avec $\underline{t_{ab}^k}$ la valeur de la borne d'incompatibilité sur le $k^{ième}$ intervalle lorsque $r_i = a$ et $r_j = b$, de même pour $\overline{t_{ab}^k}$.

Ce type de contrainte spécifie les conséquences d'un arbitrage d'un conflit. L'exemple suivant précise le fonctionnement de cette contrainte.

Exemple d'utilisation des contraintes d'incompatibilités
La figure 3.6 représente le cas où deux trains sont en conflit, les intervalles d'incompatibilité déduits sont $]-25, 30[$, le train 1 entre dans l'infrastructure à l'instant 10. La contrainte disjonctive représentant ce conflit est donc :

$$st_1 - st_2 \leq \underline{t_{12}} \vee (st_1 - st_2) \geq \overline{t_{12}}$$

À partir de la date d'entrée du train 1 ($st_1 = 10$), cette contrainte permet de déduire un nouveau domaine de validité pour la date d'entrée du train 2 (variable st_2) de la façon suivante :

$$st_2 \geq st_1 - \overline{t_{12}} \vee st_2 \leq st_1 - \underline{t_{12}}$$
$$\text{Soit } st_2 \geq 10 - (-25) \vee st_2 \leq 10 - 30$$
$$st_2 \geq 35 \vee st_2 \leq -20$$

Ainsi le train 2 ne peut entrer dans l'infrastructure dans l'intervalle de temps $]-20,\ 35[$, ce qui est représenté sur la figure 3.6 par le rectangle hachuré.

Contraintes sur les symétries

Les symétries divisent l'ensemble des affectations possibles en classes d'équivalences. L'espace de recherche décrit par les deux types de contraintes précédentes comporte des symétries,

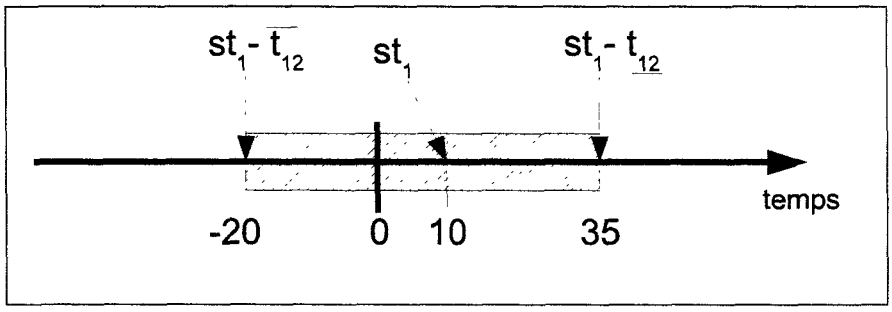

FIG. 3.6 – Représentation graphique de la contrainte d'incompatibilité

c'est-à-dire que plusieurs solutions aux caractéristiques identiques (cf. section 2.3) peuvent être obtenues, comme le montre l'exemple suivant.

Exemple de symétrie
Prenons la solution partielle suivante :

- le train 1 prend le parcours $r_i = a$ et entre dans l'infrastructure en $st_1 = 10s$,
- le train 2 le parcours $r_j = b$ et entre dans l'infrastructure en $st_2 = 15s$.

Alors la solution partielle suivante est totalement équivalente :

- le train 1 prend le parcours $r_i = b$ et entre dans l'infrastructure en $st_1 = 15s$,
- le train 2 le parcours $r_j = a$ et entre dans l'infrastructure en $st_2 = 10s$.

Ces solutions symétriques peuvent être évitées grâce à l'ajout d'une contrainte de succession des trains :

$$st_i \leq st_{i+1}, \forall i \in T, i < N - 1 \tag{3.8}$$

L'introduction de cette contrainte au modèle implique également la réduction du nombre de contraintes d'énumération (3.4) et d'incompatibilité (3.6) à poser . En effet, grâce à cette contrainte de succession (et d'ordonnancement) des trains, il n'est plus utile de poser ces deux types de contraintes pour les couples de trains (i, j) tels que $i > j$. Ainsi le nombre à poser de ces contraintes passe de n^2 à $n(n-1)/2$.

Il est à noter que l'ajout de cette contrainte n'implique pas l'annulation de toutes les disjonctions pour les contraintes d'incompatibilité entre couples de trains (3.6). En effet, les valeurs pour les variables bornes d'incompatibilité $\overline{t_{ij}^k}$ peuvent être négatives (cf. section 3.1.4), cela indiquant que le train i doit entrer dans l'infrastructure dans un laps de temps proche du train j, temps borné par $\overline{t_{ij}}$.

3.2.3 Critère

Le critère adopté pour répondre au problème de saturation d'une infrastructure ferroviaire est la minimisation de la date d'entrée du dernier train dans l'infrastructure. Ce critère est proche de la minimisation du makespan d'un problème d'ordonnancement [Esquirol et al.97].

En ce qui concerne la modélisation en un problème de Set Packing proposée par Delorme [Delorme03], le critère choisi était, comme dans la définition 3, de maximiser le nombre de trains entrant dans l'infrastructure sur un horizon temporel donné. Dans notre étude, nous ne considérons plus un temps fixé a priori mais un nombre de trains N fixés. Ces deux critères sont très voisins, en effet :

- Pour le critère considéré par Delorme, prenons une solution optimale ayant pour valeur de critère nb^* trains sur un horizon temporel fixé de ht_{fixe} secondes. Soit N_1^* représentant l'ensemble des trains circulant sur l'intervalle $[0, ht_{fixe}]$ ($nb^* = card(N_1^*)$), alors

$$\forall\, train \notin N_1^* telque\ st_{train} \notin [0, ht_{fixe}] \tag{3.9}$$

  Considérons maintenant la minimisation de la date d'entrée du dernier train (notre critère). Les trains de la solution optimale appartiennent à un autre ensemble N_2 et tel que $card(N_2) = nb^* + 1$, nous avons alors :

$$\exists\, train \in N_2\ telque\ st_{train} \notin [0, ht_{fixe}] \tag{3.10}$$

  Il n'est donc pas possible de trouver une solution ayant pour valeur d'entrée du dernier train inférieure à ht_{fixe} pour un problème à $nb^* + 1$ trains.
- De même pour le critère considérant la minimisation de la date d'entrée du dernier train : si une solution optimale a pour valeur de critère ht^* sur un problème à nb_{fixe} trains, alors pour le problème proposé par Delorme, la solution optimale sur un horizon temporel fixé à ht^* sera égale à nb_{fixe}.

Ainsi ces 2 critères sont très proches.

On peut également noter que la minimisation de la date d'entrée dans l'infrastructure du dernier train permet d'éviter l'emploi de variables supplémentaires telles que des variables booléennes, et aussi de considérer dans le critère une variable directement en relation avec les contraintes du problème.

Le critère choisi est donc modélisé comme suit :

$$\min \max_{i \in T}(st_i). \tag{3.11}$$

La contrainte de suppression des symétries (3.8) implique la succession des trains, nous avons alors : $\max_{i \in T}(st_i) = st_N$, et ainsi le critère peut s'écrire :

$$\min(st_N). \tag{3.12}$$

Le modèle général vient d'être présenté, toutefois, plusieurs améliorations peuvent être intégrées à ce modèle. Ainsi, la section suivante présente les modifications proposées pour cette

modélisation.

3.3 Améliorations

Les améliorations apportées à la modélisation permettent de réduire le nombre de contraintes posées ainsi que le nombre de variables nécessaires à la modélisation, des coupes sont proposées ainsi que l'élaboration d'une borne supérieure au problème afin de simplifier le processus de résolution. La contrainte disjonctive (3.6) est ré-écrite et enfin une contrainte supplémentaire permet d'éliminer un autre type de solution symétrique.

3.3.1 Réduction du nombre de contraintes et de variables

Les contraintes 3.4 et 3.5 portent sur tous les couples de trains de l'ensemble T. Si l'on considère tous les couples de trains, le nombre de contraintes posées est : $\frac{card(T)\times(card(T)-1)}{2}$.

Lorsque le nombre de trains est important par rapport à la durée des parcours de l'infrastructure, plusieurs contraintes sont inutiles. En effet, le temps d'occupation de l'infrastructure par un train est borné, pour le problème traité, les trains circulent en voie libre et à vitesse donnée sur l'infrastructure. Soit bt_{inc} cette borne, alors par définition, tout train i n'est plus sur l'infrastructure au delà de $st_i + bt_{inc}$. Par conséquent, les trains ayant leur date d'entrée supérieure à $st_i + bt_{inc}$ ne peuvent pas être en conflit avec le train i. La borne bt_{inc} est donc une "borne d'incompatibilité" entre trains.

La pose des contraintes d'incompatibilités avec les trains entrants après cette borne est inutile. La valeur de la borne d'incompatibilité en fonction des intervalles d'incompatibilités est donnée par :

$$bt_{inc} = \max_{bt \in \underline{I} \cup \overline{I}} (bt).$$

Cette borne n'est pas utilisable telle quelle dans la modélisation car les dates d'entrées sont des variables du problème. Pour utiliser la borne bt_{inc}, il convient de transformer cette valeur temporelle en écart d'indices sur la séquence des trains T ordonnés par la contrainte 3.8. Dans ce but, la borne bt_{inc} est transcrite en nombre maximum de trains pouvant passer dans l'infrastructure durant l'intervalle $[0, bt_{inc}]$. La transcription passe donc par la résolution du problème d'optimisation suivant :

$$\begin{cases} \max(N_{inc}) & \\ s/c & \\ enum(r_i,\ r_j,\ \underline{t^1_{ij}},\ \overline{t^1_{ij}}), & \forall i,j \in T^2, i<j, \\ ((st_i - st_j) \leq \underline{t^1_{ij}} \vee (st_i - st_j) \geq \overline{t^1_{ij}}), & \forall i,j \in T^2, i<j, \\ (r_i = a) \wedge (r_j = b) \rightarrow ((st_i - st_j) \leq \underline{t^k_{ab}} \vee (st_i - st_j) \geq \overline{t^k_{ab}}), & \forall i,j \in T^2, \\ & \forall k = 2,\ldots,card(K^{a,b}), i<j, \\ st_i \leq st_{i+1}, & \forall i \in T, i < N-1. \\ N_{inc} = \sum_{i=1}^{N} s_i & \end{cases} \tag{3.13}$$

avec $s_i = 1$ si le train i passe avant bt_{inc} ($st_i \leq bt_{inc}$), 0 sinon.

Soit N^*_{inc} la solution optimale de ce problème, les contraintes telles que $j > i + N^*_{inc}$ peuvent être supprimées. En effet, soit deux trains i et j telles que $j > i + N^*_{inc}$:

- Les deux trains ne sont pas en même temps sur l'infrastructure, ils ne peuvent donc pas être en conflit. Les contraintes d'incompatibilités (contraintes 3.6) entre ces deux trains sont de ce fait toujours satisfaites ($st_i - st_j \geq \overline{t^k_{ij}}$ est toujours vérifié). Il est alors inutile de poser les contraintes d'incompatibilités entre ces deux trains.
- Les contraintes d'énumération (contraintes 3.4) entre ces deux trains ne sont donc pas utiles, car il n'existe pas de variable « borne d'incompatibilité » à déterminer.

Exemple Occupation de l'infrastructure
Soit une infrastructure où :

- $bt_{inc} = 350$ secondes,
- $N^*_{inc} = 8$ trains.

Alors, tous les couples de trains ayant un écart d'indice de plus de 8 trains ne peuvent être en conflit, comme le montre le diagramme de Gantt de la figure 3.7.
Ce diagramme représente l'occupation de l'infrastructure par les trains, l'axe des abscisses correspond au temps d'occupation des différents trains représentés par les rectangles sur l'axe des ordonnées. On peut donc voir que si les trains ont 8 trains d'écart, leurs dates d'entrée auront au minimum 350s d'écart et donc toute contrainte d'incompatibilité sera obligatoirement vérifiée.

3.3.2 Obtention de bornes

Le principe général des méthodes de séparation et évaluation est de :

- Décrire le problème sous la forme d'une arborescence, chaque nœud correspond à une solution partielle du problème, les nœuds terminaux représentent une affectation réalisable de

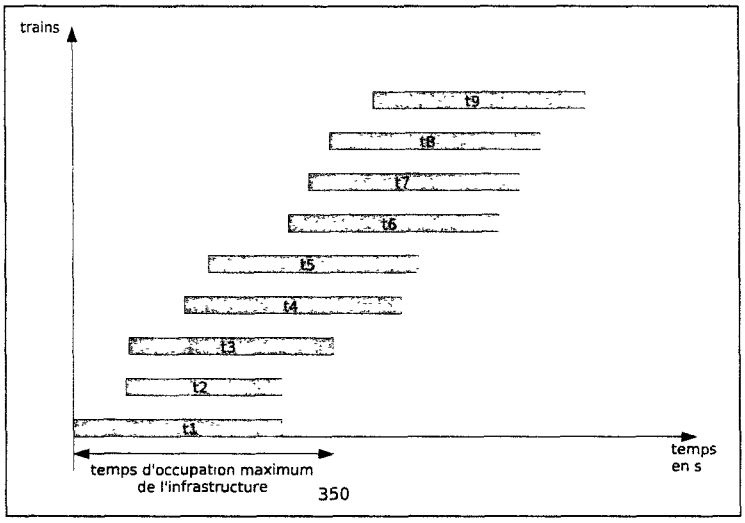

FIG. 3.7 – Diagramme de Gantt de l'occupation minimale de l'infrastructure par 8 trains

toutes les variables, soit une solution, appelée "solution admissible".
- Définir une règle de parcours de l'arbre de recherche.
- Utiliser une fonction d'évaluation des nœuds, afin de parcourir l'arbre de la meilleure manière, en élaguant des sous-branches.

Les bornes obtenues permettent d'améliorer la fonction d'évaluation et ainsi d'élaguer au plus tôt des sous-branches de l'arbre. Considérons un problème de minimisation dont une borne inférieure vaut b_{inf} et une borne supérieure b_{sup}. Lors du parcours de l'arbre, à chaque nœud une évaluation inférieure e_{inf} et supérieure e_{sup} sont calculées, en considérant les affectations effectuées. Ces valeurs de bornes obtenues permettent d'évaluer la valeur de la meilleure solution qu'il est possible d'obtenir à partir de ce nœud :

- la solution optimale aura une valeur sur le critère comprise entre b_{inf} et b_{sup},
- la valeur de la meilleure solution dans les nœuds-fils sera comprise entre e_{inf} et e_{sup}.

Il est à noter que b_{sup} et e_{sup} représentent des solutions, réalisables ou non, dont la qualité est obligatoirement aussi bonne ou plus mauvaise que la solution optimale. De ce calcul découlent trois possibilités :

- $e_{inf} > b_{sup}$ alors les nœuds-fils ne contiennent pas la solution optimale, le sous-arbre associé à ce nœud peut donc être élagué.
- $e_{sup} < b_{sup}$ alors la borne supérieure est mise à jour : $b_{sup} = e_{sup}$, cette dernière indique l'amélioration de la probabilité de la qualité des nœuds-fils.
- Sinon aucune déduction ne peut être faite, il faut parcourir les nœuds fils.

Dans ce but, des coupes, représentant une borne inférieure, ainsi qu'une borne supérieure ont été proposées pour notre problème.

Définitions de coupes

Des coupes ont été définies sur les dates d'entrées des trains dans l'infrastructure (les variables st_i) qui, par propagation, amènent à une borne inférieure du critère (la variable st_N).

Dans le problème traité, la solution recherchée est une séquence de N trains ordonnés suivant les variables st_i. C'est la contrainte de succession des trains (3.8) qui impose l'ordre chronologique dans la séquence des trains.

L'idée développée pour définir une coupe est de considérer des sous-problèmes dont la solution optimale impose un écart minimum sur les dates d'entrée de sous-séquences de la séquence des N trains ordonnés.

En considérant une sous-séquence T_s de s trains ($s < N$), et le problème de saturation $\mathcal{P}_s$ associé suivant :

$$\left[\begin{array}{ll} \min(st_s) & \\ s/c & \\ enum(r_i, r_j, \underline{t_{ij}}, \overline{t_{ij}}), & \forall i,j \in {T_s}^2, i<j, \\ & k = 1, ..., card(K_s^{r_i,r_j}). \\ ((st_i - st_j) \leq \underline{t_{ij}} \vee (st_i - st_j) \geq \overline{t_{ij}}), & \forall i,j \in {T_s}^2, i<j, \\ (r_i = a) \wedge (r_j = b) \rightarrow ((st_i - st_j) \leq \underline{t^k_{ab}} \vee (st_i - st_j) \geq \overline{t^k_{ab}}), & \forall i,j \in T^2, i<j, \\ & \forall k = 2, \ldots, card(K^{a.b}), \end{array}\right. \tag{3.14}$$

Pour chaque taille de sous-séquence s, la résolution du problème $\mathcal{P}_s$ correspondant, permettra la définition d'une série de coupes supplémentaires.

Si l'on note m^*_s la solution d'un problème $\mathcal{P}_s$, on peut remarquer que m^*_s est aussi l'écart minimum entre les dates d'entrées de tout couple de trains dont l'écart d'indice est s dans la séquence des N trains. Autrement dit, pour tout couple de trains $(i, i+s)$ de T, le train $i+s$ entrera dans l'infrastructure au moins m^*_s seconde(s) après le train i. Soit S la taille maximum des problèmes $\mathcal{P}_s$ pouvant être résolus en un temps raisonnable. Une fois les valeurs de m^*_s, $s = 2, \ldots, S$ calculées, les coupes suivantes peuvent être définies :

$$st_i + m^*_s \leq st_{i+s} \quad s = 2, \ldots, S, i = 1, \ldots, N - s \tag{3.15}$$

Le paragraphe suivant présente un exemple sur les intervalles de succession des trains :

Exemple Intervalles de succession des trains

Supposons que la valeur minimale du makespan pour une séquence de 4 trains est de 15 secondes, il sera alors impossible de trouver une sous-séquence de quatre trains dans un intervalle inférieur à ces 15 secondes pour une séquence de N trains ($N \geq 4$). La contrainte $st_{i+3} \geq st_i + 15$, $\forall i \in 1, \ldots, N-3$ peut dès lors être ajoutée au problème.

Ainsi ces contraintes sont obtenues par la résolution de plusieurs problèmes de petites tailles.

Soit N_s la taille maximale des sous-séquences de trains considérées, soit m_s^* la valeur optimale du makespan du problème à s trains, les coupes suivantes peuvent être ajoutées au modèle :

$$st_{i+s} \geq st_i + m_s^*, \forall i \leq N - s, s = 2, \ldots, N_s \tag{3.16}$$

Ce type de contrainte permet d'avoir une borne inférieure à notre problème par propagation de st_1 à st_N. En effet, nous avons les contraintes suivantes :

$$\begin{aligned} st_1 + m_s^* &\leq st_s \\ st_s + m_s^* &\leq st_{2\times s} \\ st_{2\times s} + m_s^* &\leq st_{3\times s} \\ &\ldots \\ st_{(\lfloor N/s \rfloor - 1)\times s} + m_s^* &\leq st_{\lfloor N/s \rfloor \times s} \\ \Rightarrow \quad st_1 + \lfloor N/s \rfloor \times m_s^* &\leq st_{\lfloor N/s \rfloor \times s} \end{aligned}$$

Ainsi, la variable st_1 est reliée à st_{N_s} par ce type de contrainte. De même la variable st_{N_s} est reliée à la variable $st_{2\times N_s}$, ..., ainsi de suite jusque st_N. Par propagation nous avons donc une relation entre la variable st_1 et la variable st_N, ce qui revient à une borne inférieure.

Ce type de contrainte domine également la contrainte de suppression des symétries (3.8) étant donné que $m_s^* \geq 0$.

Une borne supérieure

L'obtention d'une borne supérieure à un problème de minimisation peut permettre de le résoudre plus rapidement. Ainsi pour obtenir une borne supérieure de notre problème $\mathcal{P}$, la résolution d'un problème $\mathcal{P}'$ est effectuée. Ce problème $\mathcal{P}'$ est plus contraint que $\mathcal{P}$ et tel que $sol(\mathcal{P}') \subseteq \int\!\!\updownarrow(\mathcal{P})$ où $sol(\mathcal{P})$ regroupe l'ensemble des solutions du problème $\mathcal{P}$.

La méthode proposée peut se décrire en deux étapes :

1. Rechercher une bonne solution (voire une solution optimale) du problème $\mathcal{P}$ de saturation sur une sous-séquence de trains de taille n_{seq} ($n_{seq} < N$),
2. Juxtaposer autant de fois qu'il le faut la solution du sous-problème de l'étape précédente afin de construire une solution valide du problème sur la séquence de trains de taille N.

Il est clair que l'application directe de ces deux étapes se heurte à la vérification des contraintes de compatibilité entre les trains des sous-séquences juxtaposées. En effet, si l'étape 1 garantit bien

que le parcours et la date d'entrée d'un train i seront compatibles avec un train $i+1$, dans l'étape 2, la compatibilité entre le train i et le train $i+1+n_{seq}$ n'est pas assurée.

Pour pallier à cette difficulté, la mise en œuvre employée consiste à partir d'un problème à N trains (solution valide de l'étape 2) et à y ajouter des contraintes qui réduisent l'ensemble des variables de décision de telle façon que sa résolution soit équivalente à la résolution d'un problème à n_{seq} trains (étape 1). Avec les contraintes ajoutées, on doit passer d'un problème $\mathcal{P}$ à N trains où il faut trouver N affectations d'itinéraires et dates d'entrée à un problème $\mathcal{P}'$ toujours à N trains mais où il faut trouver n_{seq} affectations d'itinéraires et dates d'entrée.

Les contraintes ajoutées au problème initial P à N trains pour former le problème P' sont les suivantes :

$$r_i = r_{i+n_{seq}}, \quad \forall i = 1, \ldots, N - n_{seq} \tag{3.17}$$

$$st_{i+n_{seq}} = st_i + st_{n_{seq}}, \quad \forall i = 1, \ldots, N - n_{seq}. \tag{3.18}$$

Les contraintes 3.17 imposent que toutes les séquences de n_{seq} trains prennent une même séquence de parcours. De même pour les variables dates d'entrée, les contraintes 3.18 permettent d'avoir le même écart entre les dates d'entrée des trains de chaque séquence. Avec ces deux types de contraintes, on passe d'un problème où il faut déterminer $2 \times N$ variables parcours et dates d'entrée à un problème $\mathcal{P}'$ à $2 \times n_{seq}$ variables, le problème $\mathcal{P}'$ gardant les mêmes contraintes que $\mathcal{P}$.

Le critère choisi pour le problème $\mathcal{P}'$ est de minimiser la date d'entrée du premier train de la deuxième séquence : $\min st_{n_{seq}+1}$, afin de tenir compte de l'écart de temps entre les sous-séquences n_{seq}. On note $st\_seq_i^*$ la valeur de la variable st_i dans la solution optimale de P'. La valeur optimale du critère considérée est notée : $st\_seq_{n_{seq}+1}^*$. Les contraintes de ce problème impliquent : $st\_seq_{2\times n_{seq}}^* = 2 \times st\_seq_{n_{seq}}^*$. Une fois cette valeur optimale connue, cette borne supérieure est ajoutée au problème initial P par la contrainte :

$$st_N \leq st\_seq_{n_{seq}+1}^* \times \lfloor N/n_{seq} \rfloor + st\_seq_{(N \bmod n_{seq})}^*, \tag{3.19}$$

en considérant $st\_seq_0^* = 0$.

On remarque que l'approche adoptée pour obtenir cette borne supérieure est similaire au calcul d'une grille horaire cadencée avec un cycle de n_{seq} trains.

3.3.3 Redéfinition de la contrainte d'incompatibilité

La contrainte d'incompatibilité (3.6) est une contrainte disjonctive, c'est à dire une contrainte ne permettant de propagation (élimination de valeurs de variables) que lorsque le choix de l'une ou l'autre alternative est fait. En effet, tant que le choix $st_i - st_j \leq \underline{t_{ij}}$ ou $st_i - st_j \geq \overline{t_{ij}}$ n'est pas établi, aucune information sur les valeurs supérieures et inférieures des st ne peut être déduite par

les algorithmes de propagation classique, comme le montre l'exemple suivant :

Exemple Non propagation sur les contraintes inc_1
Soit la contrainte :

$$(st_1 - st_2) \leq -70 \vee (st_1 - st_2) \geq -20,$$

alors tant que le choix de l'une ou l'autre alternative n'est faite, aucune propagation sur les st ne peut être effectuée. C'est à dire que l'algorithme ne tiendra pas en compte le fait que $st_1 - st_2 \notin]-70, -20[$, aucune valeur des st ne sera éliminée de leur domaine.

Le but est alors de ré-écrire cette contrainte afin de pouvoir propager sans que, nécessairement, le choix de l'ordonnancement des trains ne soit fait. Pour cela, un nouvel ensemble de contraintes, nommé inc_2, est proposé :

$$\delta_{ij} \notin]\underline{t_{ij}^k}, \overline{t_{ij}^k}[, \forall i, j \in T^2, \forall k = 1, \ldots, N_k. \tag{3.20}$$

avec $N_k = \max_{i,j \in T^2} card(K^{r_i,r_j})$, K^{r_i,r_j} représentant l'ensemble des conflits entre les couples de trains empruntant les parcours r_i et r_j et avec, par convention, $st_i - st_j = \delta_{ij}, \forall i, j \in T^2$. δ_{ij} représente donc la variable d'écart entre les dates d'entrées des trains i et j dans l'infrastructure.

Pour cette contrainte, le même problème que pour la contrainte d'incompatibilité inc_1 se pose, à savoir que le nombre de domaines d'incompatibilité entre les trains n'est connu que lorsque les parcours sont affectés. Ainsi, cette contrainte d'incompatibilité s'écrit sous la forme d'une contrainte classique et d'une méta-contrainte posée sous la condition que le couple de parcours possède plus d'un conflit potentiel. Nous avons alors la modélisation de cette contrainte d'incompatibilité :

$$\begin{aligned} &\delta_{ij} \notin]\underline{t_{ij}^1}, \overline{t_{ij}^1}[, \forall i, j \in T^2, i > j, \\ &(r_i = a) \wedge (r_j = b) \rightarrow (\delta_{ij} \notin]\underline{t_{ab}^k}, \overline{t_{ab}^k}[), \\ &\forall i, j \in T^2, i > j, \ \forall k = 2, \ldots, card(K^{a,b}), \end{aligned} \tag{3.21}$$

avec $\underline{t_{ab}^k}$ la valeur de la borne d'incompatibilité sur le $k^{ième}$ intervalle lorsque $r_i = a$ et $r_j = b$, de même pour $\overline{t_{ab}^k}$.

Cette contrainte utilise les techniques de propagation sur intervalles. En considérant $inf(x)$ la valeur minimale du domaine de la variable x et $sup(x)$ sa valeur maximale, les mécanismes de propagation de cette contrainte sont les suivants :

- Si $inf(\overline{t_{ij}}) > sup(\delta_{ij})$, alors on a $inf(\underline{t_{ij}}) = inf(\delta_{ij})$ et $sup(\delta_{ij}) = sup(\underline{t_{ij}})$. δ_{ij} est définitivement à gauche de l'intervalle ($st_i - st_j \leq \underline{t_{ij}}$).
- Si $sup(\underline{t_{ij}}) < inf(\delta_{ij})$, alors on a $sup(\overline{t_{ij}}) = sup(\delta_{ij})$ et $inf(\delta ij) = inf(\overline{t_{ij}})$. δ_{ij} est définitivement à droite de l'intervalle($st_i - st_j \geq \overline{t_{ij}}$).

- Éliminer toutes les valeurs de $\delta_{ij} \in [sup(\underline{t_{ij}}), inf(\overline{t_{ij}})]$.
- Éliminer toutes les valeurs de $\underline{t_{ij}} \notin dom(\delta_{ij})$.
- Éliminer toutes les valeurs de $\overline{t_{ij}} \notin dom(\delta_{ij})$.

Cette contrainte permet de propager depuis les variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$ vers les variables st_i, st_j, mais également depuis les variables st_i, st_j vers les variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$. Ceci implique ensuite, grâce à la contrainte d'énumération (3.4) de propager les réductions de domaines des variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$ vers les variables parcours r_i et r_j.

Il est également à noter que grâce à la contrainte de suppression des symétries (3.8), le domaine des variables δ_{ij} est borné en valeur supérieure à 0.

3.3.4 Symétries sur les parcours

La contrainte de suppression des symétries (3.8) ne couvre pas toutes les symétries du modèle présenté. Cette contrainte évite la plupart des solutions équivalentes obtenues lors de l'affectation des variables st_ir_i, st_jr_j de deux trains d'indices i et j. Plus précisément, elle supprime les solutions qui consistent à intervertir les valeurs des variables du train i avec celles du trains j.

L'opérateur d'ordre de la contrainte (3.8) n'est pas une inégalité stricte, ce qui conduit à la persistance de symétries pour les cas où les variables dates d'entrées prennent des valeurs identiques. En effet, si pour deux trains i, j, $st_i = st_j$, la contrainte (3.8) n'empêchera pas de considérer les deux solutions équivalentes où les valeurs des parcours sont interverties.

Exemple Symétries sur les parcours

Considérons une solution partielle avec trois trains dont l'affectation partielle est la suivante :

trains	1		2		3	
variables	st_1	r_1	st_2	r_2	st_3	r_3
domaines	0	3	0	5	$\mathbb{N}$	R

Quelles que soient les affectations des variables relatives au train 3, cette solution est équivalente à :

trains	1		2		3	
variables	st_1	r_1	st_2	r_2	st_3	r_3
domaines	0	5	0	3	$\mathbb{N}$	R

Pour supprimer ce type de symétrie, l'idée est d'ajouter au modèle des valeurs de variables interdites, à chaque fois que ce type de solutions (partielles ou non) est obtenu durant le processus de résolution.

Pour ce genre de contrainte, la notion de no-good (cf. [Schiex et al.94] et [Dechter90]) est utilisée, un no-good correspond à un ensemble d'affectations de valeur à des variables qui ne peuvent donner de solution au problème ou qui ne sont pas souhaitées.

Dans notre cas, les no-goods ajoutés sont les solutions partielles symétriques aux solutions partielles trouvées et qui comportent des dates d'entrées identiques. Les no-goods contiennent l'ensemble des valeurs fixées pour les parcours et dates d'entrées de la solution partielle.

Ces solutions symétriques sont supprimées à l'aide d'une contrainte globale qui interdit toute permutation de valeurs associées aux trains entrant dans l'infrastructure en même temps. Cette contrainte suit, en quelque sorte, le même principe que la contrainte d'énumération (3.4), si ce n'est qu'il ne s'agit plus de tuples admissibles mais de no-good et que ces no-goods sont ajoutés au fur et à mesure des affectations déjà effectuées. Par la suite, cette contrainte sera nommée :

$$NGBS(r, st), \forall r_i, st_i | st_i \textit{fixé}, \exists j | st_j = st_{j+1}. \tag{3.22}$$

Sur l'exemple détaillé précédemment, la contrainte $NGBS(r, st)$ agit de la manière suivante : si $r_1 = 5$, $st_1 = 0$ et $st_2 = 0$ alors l'affectation $r_2 = 3$ est interdite.

3.4 Modèles et spécificités des problèmes traités

À partir des éléments de modélisation précédents, plusieurs modèles ont été déclinés afin de pouvoir évaluer l'impact des améliorations proposées. Nous avons présenté cinq types d'améliorations, que nous noterons : 1, 2, 3, 4 et 5, ces cinq dénominations désignent :

1. l'amélioration permettant d'éviter les solutions symétriques (cf. section 3.3.4), elle correspond à l'ajout de la contrainte :

$$NGBS(r, st), \forall r_i, st_i | st_i \textit{fixé}, \exists j | st_j = st_{j+1}.$$

2. la redéfinition de la contrainte d'incompatibilité disjonctives inc_1 en une contrainte sur des ensembles inc_2 3.3.3), soit les contraintes inc_2 dont l'équation est :

$$\begin{aligned} \delta_{ij} \notin]\underline{t^1_{ij}}, \overline{t^1_{ij}}[, \forall i, j \in T^2, i > j, \\ (r_i = a) \wedge (r_j = b) \rightarrow (\delta_{ij} \notin]\underline{t^k_{ab}}, \overline{t^k_{ab}}[), \\ \forall i, j \in T^2, i > j, \ \forall k = 2, \ldots, card(K^{a,b}), \end{aligned}$$

3. la réduction du nombre de contraintes et de variables grâce au temps d'occupation maximum de l'infrastructure par un train (cf. section 3.3.1), cette amélioration se traduit par l'élimination de toutes les contraintes intervenant entre les trains éloignés de plus de N^*_{inc},

4. l'ajout de coupes sur les variables dates de départs (cf. section 3.3.2), ces coupes sont ajoutées par l'intermédiaire des contraintes de type :

$$st_{i+s} \geq st_i + m_s^*, \forall i \leq N - s, s = 2, \ldots, N_s,$$

5. l'ajout de la borne supérieure (cf. section 3.3.2) est utilisé par l'intermédiaire d'une contrainte dans le modèle :

$$st_N \leq st\_seq^*_{n_{seq}+1} \times \lfloor N/n_{seq} \rfloor + st\_seq^*_{(N \bmod n_{seq})}.$$

Chacune de ces améliorations a été ajoutée au modèle initial M_0 décrit par l'équation 3.23 :

$$\left[\begin{array}{ll}
\min(st_N) & \\
s/c & \\
\quad \textit{enum}(r_i, r_j, \underline{t_{ij}}, \overline{t_{ij}}), & \forall i,j \in T^2, i < j \\
\quad st_i - st_j \leq \underline{t_{ij}} \vee st_i - st_j \geq \overline{t_{ij}}, & \forall i,j \in T^2, i < j \;\; //inc_1 \\
\quad (r_i = a) \wedge (r_j = b) \rightarrow st_i - st_j \leq \underline{t^k_{ab}} \vee st_i - st_j \geq \overline{t^k_{ab}}, & \forall i,j \in T^2, i < j \\
\qquad \forall k = 2, \ldots, card(K^{a,b}) //inc_1 & \\
\quad st_i \leq st_{i+1} & \forall i \in T // \textit{suppression symétries}
\end{array}\right. \tag{3.23}$$

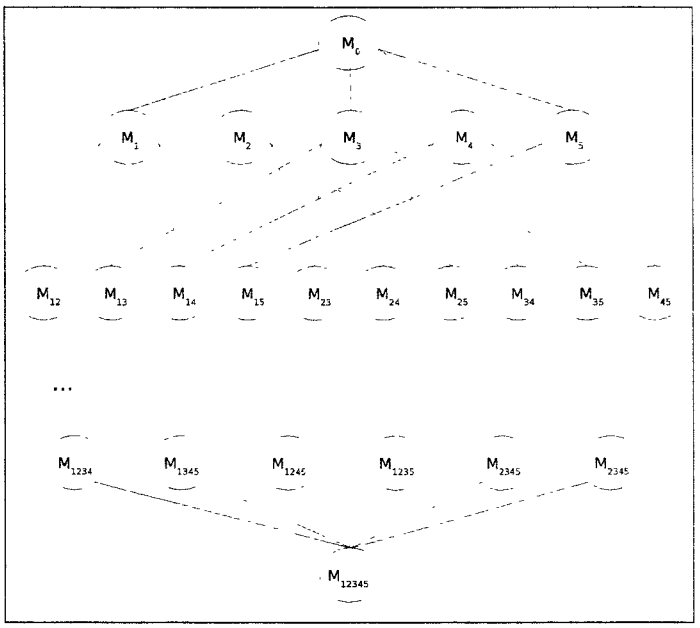

FIG. 3.8 – Treillis des modèles

Nous avons alors le treillis décrit par la figure 3.8 partant du modèle initial, sans amélioration M_0 et auquel est ajouté à chaque niveau une amélioration. Ainsi M_{1234} désigne le modèle initial auquel les améliorations 1, 2, 3 et 4 ont été ajoutées, soit les contraintes permettant d'éliminer les solutions symétriques, la redéfinition de la contrainte d'incompatibilité, la réduction du nombre de contraintes et variables et l'ajout des coupes. Le modèle final, comportant l'ensemble des améliorations est noté M_{12345}, ce modèle correspond à l'équation 3.24.

$$
\left[\begin{array}{ll}
\min(st_N) & \\
s/c & \\
\quad \textbf{\textit{enum}}(r_i, r_j, \underline{t_{ij}}, \overline{t_{ij}}), & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}} \\
\quad \delta_{ij} \notin]\underline{t_{ij}}, \overline{t_{ij}}[, & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}} \ \ //inc_2 \\
\quad (r_i = a) \wedge (r_j = b) \rightarrow \delta_{ij} \notin]\underline{t^k_{ab}}, \overline{t^k_{ab}}[, & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}}, \\
\qquad \forall k = 2, \ldots, card(K^{a,b}) & \\
\quad st_{i+s} \geq st_i + m^*_s, & \forall i \leq N-s, s = 2, \ldots, N_s \\
\quad st_N \leq st^*_{n_{seq}+1} \times \lfloor N/n_{seq} \rfloor + st\_seq^*_{N \bmod n_{seq}} & \\
\quad NGBS(r, st), & \forall r_i, st_i | st_i \textit{fixé}, \exists j | st_j = st_{j+1}.
\end{array}\right. \tag{3.24}
$$

Lors des phases de tests le modèle M_{12345} a été pris comme modèle de référence initial afin de tester l'impact de chacune des améliorations proposées. À chaque test, le modèle de référence est actualisé par le meilleur des 2 modèles testés, soit :

$$
\begin{aligned}
&M^0_{ref} = M_{12345} \\
&M^1_{ref} = \min(M^0_{ref}, M_{2345}) \Leftrightarrow \min(M^0_{ref}, M^0_{ref/1}) \\
&M^1_{ref} = \min(M^0_{ref}, M_{1345}) \Leftrightarrow \min(M^0_{ref}, M^0_{ref/2}) \\
&\ldots
\end{aligned}
$$

où $M^0_{ref/1}$ correspond au modèle de référence sans l'amélioration 1.

Ce qui nous donne la relation de récurrence suivante, et pour l'itération i :

$$
M^i_{ref} = \min(M^{i-1}_{ref}, M^{i-1}_{ref/(i \mod 5)}) \tag{3.25}
$$

où mod correspond au modulo de la fraction $\frac{i}{5}$.

3.5 Données et spécificités des problèmes traités

L'évaluation du modèle et des algorithmes de résolution se base sur des instances réelles (provenant d'une infrastructure ferroviaire) et aléatoires pour tester l'algorithme sur des problèmes plus variés. Un total de 18 type d'instances est proposé, ces différents types correspondant à des

infrastructures et type de trains pouvant y circuler. Une instance sera caractérisée par un type et un nombre de trains. Le format spécifique de ces données est par la suite expliqué.

3.5.1 Données réelles

D'un côté nous avons les instances réelles. Les données ont été collectées depuis le nœud de Pierrefitte-Gonnesse où circule un ensemble hétérogène de trains. La figure 3.9 en présente une vue schématique (plusieurs voies, voies en double sens, croisements).

Les valeurs d'incompatibilités sur l'infrastructure ont été obtenues grâce à une phase de simulation depuis le simulateur SISYFE [Fontaine et al.01]. Un très grand nombre d'instances peuvent être testé une fois l'ensemble des valeurs d'incompatibilités obtenues, nous nous sommes limités à 3 types représentant un large panel des combinaisons possibles :

- celle considérant la circulation des TGV et trains de marchandises uniquement, dénommée TGV-MA, l'ensemble des combinaisons de parcours et types de trains possibles est de 19 ($card(R) = 19$),
- celle considérant la circulation des trains corails et trains de marchandises uniquement, dénommée CL-MA, l'ensemble des combinaisons de parcours et types de trains possibles est de 26 ($card(R) = 26$),
- celle considérant l'ensemble des parcours et types de trains, dénommée TGV-CL-MA, où l'ensemble des parcours admissibles par les TGV, trains corails et trains de marchandises sont représentés, soit un total de 37 combinaisons d'itinéraires et types de trains possibles ($card(R) = 37$).

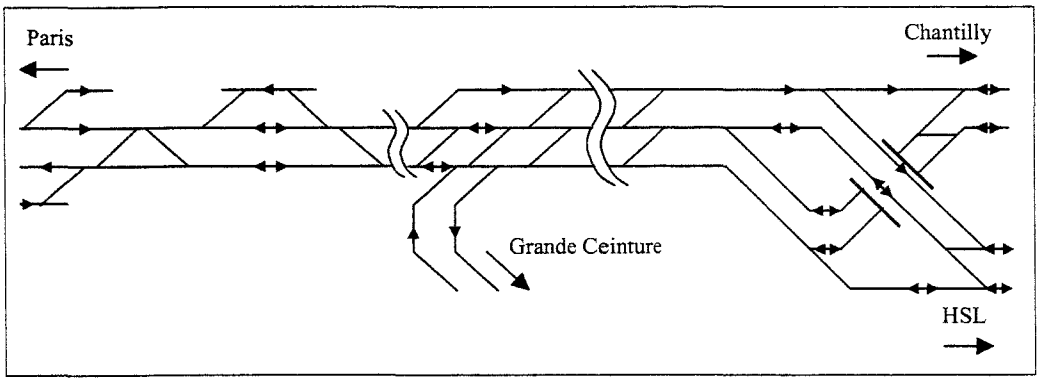

FIG. 3.9 – Nœud de Pierrefitte-Gonnesse

3.5.2 Données aléatoires

D'un autre côté, 15 instances aléatoires ont été générées. Leurs caractéristiques sont semblables aux instances réelles, à savoir : même probabilité que 2 trains empruntent un tronçon commun

itinéraire et type de train		bornes d'incompatibilité	
r_i	r_j	$\underline{t_{ij}}$	$\overline{t_{ij}}$
A	A	-68	68
A	B	-250	-15
A	B	-100	150
A	C	-120	-10
B	B	-98	98
C	C	-79	79

TAB. 3.9 – Format des données

(90%), même ordre de grandeur pour le temps maximum d'incompatibilité ($bt_{inc} \leq 500$) et temps de succession identique pour 2 trains de même type et empruntant le même parcours (de l'ordre de $80s$). Ce qui nous donne :

- 5 instances nommées alea1 correspondant à 19 combinaisons d'itinéraires et types de trains possible ($card(R) = 19$),
- 5 instances nommées alea2 correspondant à 26 combinaisons d'itinéraires et types de trains possible ($card(R) = 26$),
- 5 instances nommées alea3 correspondant à 37 combinaisons d'itinéraires et types de trains possible ($card(R) = 37$),

3.5.3 Format des données

Les données nécessaires à l'utilisation des modèles présentés correspondent aux valeurs possibles pour les variables d'incompatibilité associées aux parcours et types de trains. Ces données donnent les tuples admissibles pour la contrainte d'énumération (3.4). Le tableau 3.9 présente le format des données pour une infrastructure présentant trois possibilités de combinaison d'itinéraires et de types de trains : A, B et C.

- La ligne correspondant au choix de parcours pour un couple de train (A, A) signifie que si $r_i = A$ et $r_j = A$ alors $\underline{t_{ij}} = -68$ et $\overline{t_{ij}} = 68$. Les deux trains (empruntant le même parcours A) doivent se suivre avec plus de 68 secondes d'écart.
- Le cas des choix de parcours (A, B) implique deux conflits entre les trains i et j, ainsi 2 couples de variables bornes d'incompatibilité seront nécessaires à la modélisation. Le premier couple prenant les valeurs $\underline{t^1_{ij}} = -250$ et $\overline{t^1_{ij}} = -15$, le second les valeurs $\underline{t^2_{ij}} = -100$ et $\overline{t^2_{ij}} = 150$.
- Il n'existe pas de ligne correspondant au choix $r_i = B$ et $r_j = C$, ce qui indique que ces deux parcours n'ont pas de zones communes et ne peuvent donc pas être en conflit.
- Le choix $r_i = C$ et $r_j = A$ implique que les variables bornes d'incompatibilité doivent prendre les valeurs : $\underline{t_{ij}} = 120$ et $\overline{t_{ij}} = 10$.

Spécificités du problème de saturation

Dans le problème de saturation traité la grille horaire initiale est considérée vide, c'est-à-dire que chaque train peut être de n'importe quel type, emprunter n'importe quel itinéraire et entrer dans l'infrastructure à n'importe quel moment de l'étude. Ainsi, les domaines des variables sont :

- Les domaines des variables parcours (r_i) sont l'ensemble R de tous les types de trains et itinéraires admissibles dans l'infrastructure.
- A fortiori les domaines des variables bornes d'incompatibilité ($\underline{t_{ij}}$ et $\overline{t_{ij}}$) sont les ensembles des valeurs permettant de gérer tous les conflits de l'infrastructure, elles sont donc incluses dans $\underline{I}$ et $\overline{I}$.
- Les domaines des variables dates d'entrée dans l'infrastructure (st_i) sont compris entre 0 et $horizon$, où $horizon$ correspond au temps de l'étude, dans notre cas $horizon$ correspond à une valeur très importante garantissant l'obtention d'au moins une solution (comme par exemple : $N*$durée maximale de traversée de l'infrastructure.

Spécificités du problème de faisabilité

Une adaptation de notre modèle peut être faite afin de résoudre le problème de faisabilité d'une grille horaire. Le problème de faisabilité, comme vu en section 1.3.3 page 27 se définit comme suit :

Définition 22 (Problème de faisabilité). *Le problème de faisabilité consiste à trouver les itinéraires d'un ensemble de trains pour que le parcours sur l'infrastructure étudiée se réalise sans retard et avec un niveau de robustesse (marge permettant de minimiser l'impact de perturbations) donné.*

Le problème de faisabilité peut donc être vu comme le problème de saturation auquel le domaine des variables est restreint :

- Les domaines des variables parcours (r_i) peuvent être restreints à un type de trains et un ensemble d'itinéraires.
- Les domaines des variables bornes d'incompatibilité ($\underline{t_{ij}}$ et $\overline{t_{ij}}$) sont alors réduits en fonction des valeurs admissibles pour les variables parcours.
- Les domaines des variables dates d'entrée dans l'infrastructure (st_i) sont réduits en fonction de la grille horaire pré-établie.

3.6 Conclusion

Ce chapitre a présenté le modèle utilisé pour résoudre le problème de saturation ferroviaire. Les conflits ont été définis en deux catégories et gérés via deux principes : espacement et enclenchement. À partir de cette définition le modèle de base a été présenté. Ce problème a été modélisé en un problème de satisfaction de contraintes, dont les spécificités ont été présentées au chapitre

précédent. Les variables utilisées sont regroupées en deux catégories : variables d'affectation et variables d'ordonnancement. Les contraintes principales permettent de faire la correspondance entre les parcours et les dates d'incompatibilité pour chaque couple de trains, mais aussi de gérer le trafic (ordonnancement des trains). Le critère proposé est de minimiser la date d'entrée du dernier train dans l'infrastructure. Suite à cela diverses améliorations portant sur la diminution du nombre de variables et contraintes, la définition de bornes et de coupes, de nouvelles contraintes ont été présentées et détaillées. Les différents modèles qui seront utilisés lors de la résolution ont été décrits ainsi que les spécificités des problèmes traités. Un résumé des principales variables et du modèle est donné en partie annexe à la fin de ce mémoire.

Le chapitre suivant présente les méthodes de résolutions de programmation par contraintes utilisées, tant au niveau de la résolution exacte (choix de variables, valeurs) que de la résolution approchée (heuristiques, méta-heuristiques, hybridation de méthodes).

Chapitre 4

Approches de résolution

La modélisation choisie dans le chapitre précédent pour le problème d'évaluation de la capacité d'infrastructures ferroviaires est celle d'un CSP temporel présenté en section 2.5. Deux types de variables ont été retenus : les variables d'affectation et les variables d'ordonnancement. Sur ces variables, trois ensembles de contraintes ont été définis : des contraintes d'énumération, des contraintes d'incompatibilités et suppression des symétries. Des améliorations ont été apportées à ce modèle : suppression de contraintes, borne supérieure et coupes.

Ce chapitre propose d'étudier les techniques employées pour la résolution de ce problème. Ces techniques portent sur le paramétrage : des heuristiques de choix de variables et de valeurs, des coupes, une borne supérieure, des méthodes de résolution exactes et approchées ...

Plusieurs améliorations du modèle ont été proposées au chapitre 3, dont certaines impliquent l'utilisation de paramètres ou la résolution de sous-problèmes particuliers. Ce chapitre présente les valeurs retenues pour les paramètres de ces améliorations, ainsi que ceux des méthodes de résolution de ces sous-problèmes.

4.1 Stratégies de résolution

Comme nous l'avons vu dans le chapitre portant sur la programmation par contraintes (cf. chapitre 2), les algorithmes de résolution s'appuient principalement sur deux types d'heuristiques : le choix de la variable à affecter et le choix de la valeur à affecter. Cette section présente les heuristiques de ce type ainsi que les expérimentations qui ont permis de les évaluer.

4.1.1 Heuristique de choix d'instanciation des variables

Ce type d'heuristique utilise une fonction guidant les ordres de variables à instancier dans l'exploration d'un arbre de recherche. Son but est de permettre de s'arrêter au plus tôt dans l'arbre de recherche. L'étude de la taille des domaines des variables est le critère le plus répandu pour l'ordre de sélection d'une variable. À partir de ce critère, le choix généralement utilisé consiste à

suivre une heuristique « d'échec d'abord » (« first fail »). Cela consiste à choisir la variable de plus petit domaine ou la variable la plus contrainte.

Dans le problème ferroviaire traité, il existe des dépendances fortes entre famille de variables, dépendances induites par les contraintes. Cela nous a conduit à considérer les types des variables pour les choix d'ordre d'instanciation. Chaque train est représenté par 3 types de variables (cf. 3.1.1) :

1. la variable « parcours » r_i,
2. les variables « bornes d'incompatibilités » $\underline{t_{ij}}$, $\overline{t_{ij}}$,
3. la variable « dates d'entrées » st_i,

Ces trois types de variables d'un train sont fortement liées les unes aux autres par les contraintes du problème (contraintes d'énumération (3.4) et d'incompatibilités (3.6) et (3.22)). L'affectation d'un type de variable aura donc un impact fort sur les autres types, ceci se traduisant par une propagation et donc une réduction importante de leur domaine de valeur.

En considèrant une heuristique de choix regroupant les choix de variable par leur type, il y a 6 ordres possibles d'énumération :

1. t, r, st : variables bornes d'incompatibilités ($\underline{t_{ij}}$, $\overline{t_{ij}}$) puis parcours (r_i) et enfin dates d'entrées (st_i),
2. t, st, r : variables bornes d'incompatibilités puis dates d'entrées et enfin parcours,
3. r, t, st : variables parcours puis bornes d'incompatibilités et enfin dates d'entrées,
4. r, st, t : variables parcours puis dates d'entrées et enfin bornes d'incompatibilités,
5. st, r, t : variables dates d'entrées puis parcours et enfin bornes d'incompatibilités,
6. st, t, r : variables dates d'entrées puis bornes d'incompatibilités et enfin parcours,

La contrainte d'énumération (contrainte 3.4) fait le lien entre les variables t et r. Aussi dès que l'ensemble des variables r est fixé (respectivement t), l'ensemble des t (respectivement r) l'est aussi ou tout au moins les domaines s'en trouvent très fortement réduits. Ainsi les ordres (1) et (2) sont équivalents tout comme les ordres (3) et (4).

L'énumération des st en premier n'est pas décrite ici car le domaine de cette variable est très important et ne peut être réduit qu'en connaissant les valeurs des variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$. Ceci nous a été confirmé par les premières expérimentations qui ont donné des temps de résolution exhorbitants. Nous ne traiterons donc pas les ordres (5) et (6).

Les deux possibilités retenues pour l'ordre des variables dans cette étude sont les ordres (1) et (3), notés : t-r-st et r-t-st. Ces deux ordres repoussent l'instanciation des variables dont les domaines sont les plus grands à la fin de l'exploration de l'arbre, on peut donc les considérer comme des heuristiques « d'échec d'abord ».

Ordre « t-r-st »

Les variables $\underline{t_{ij}}$ sont celles ayant une influence directe sur la succession des trains et donc sur les variables dates d'entrées st_i ce qui influe directement sur le critère.

L'ordre d'instanciation des variables $\underline{t_{ij}}$ et $\overline{t_{ij}}$ est celui impliquant des couples de trains les plus proches dans l'ordre de succession. En effet, les trains successifs ont une probabilité plus importante d'être en conflit que des trains éloignés (cf. section 3.3.1), le train $i+1$ est le successeur direct du train i. Les trains sont ordonnés selon leurs indices grâce aux coupes (cf. équation 3.16), l'ordre d'instanciation revient à l'ensemble des $\underline{t_{ij}}$, $\overline{t_{ij}}$ $\forall i = 1, \ldots, N; j = i+1, \ldots, N$.

Une fois les variables bornes d'incompatibilités affectées, le domaine des variables parcours est très fortement réduit grâce à la contrainte d'énumération (3.4), ainsi leur affectation ne nécessite pas l'utilisation d'un ordre particulier, celui sélectionné étant l'ordre des indices des trains : $r_0, r_1, \ldots, r_N$. Il en va de même pour les variables dates d'entrées : $st_0, st_1, \ldots, st_N$.

Ordre « r-t-st »

Les variables r_i sont celles ayant le plus grand impact sur les autres variables. En effet, l'affectation du parcours d'un train permet une propagation forte sur les variables bornes d'incompatibilités par la contrainte d'énumération (3.4). Le nombre et la taille des domaines de ces variables est moins important que pour les variables bornes d'incompatibilités, ainsi un effort moindre sera nécessaire pour leur affectation (moins d'itérations). Par contre, l'affectation de leurs valeurs nécessite une stratégie d'évaluation plus complexe, car on dispose de moins d'information pour guider le choix. L'instanciation des variables parcours s'effectue dans l'ordre de succession des trains, les trains successifs ayant un lien plus fort que des trains éloignés.

Une fois les parcours affectés, les domaines des variables des bornes d'incompatibilités t_{ij} sont réduits à une valeur par la contrainte d'énumération 3.4 et les variables sont donc directement affectées.

Enfin, les variables dates d'entrées sont affectées dans l'ordre croissant des i : $st_0, st_1, \ldots, st_N$.

Pour les deux heuristiques de choix qui viennent d'être présentées, lorsque l'affectation de deux parcours comporte un conflit multiple (cf. section 3.1.4), les contraintes d'incompatibilités supplémentaires (cf. partie 3.2.2) sont ajoutées juste après l'affectation des variables parcours et bien évidemment avant l'affectation des variables st_i.

Il n'y a pas de mixage d'ordre d'affectation entre les différents types de variables (affecter un r_i puis un t_{ij} puis $r_{i+1}, \ldots$), car les variables r_i et t_{ij} sont directement liées par la contrainte d'énumération.

4.1.2 Heuristique de choix de valeurs

En suivant les heuristiques présentées en section précédente 4.1.1, deux ordres de choix de valeurs ont été implémentés :

- ordre sur les valeurs pour les bornes d'incompatibilités ($\underline{t_{ij}}$ et $\overline{t_{ij}}$) et
- ordre sur les valeurs des itinéraires et type de trains (r_i).

Ordre sur les bornes d'incompatibilités

Les variables bornes d'incompatibilités permettent, a priori, d'avoir une stratégie d'énumération intéressante. En effet, le temps de succession des trains est déterminé par les intervalles d'incompatibilités (cf. section 3.2.2). On rappelle que :

- $\underline{t_{ij}}$ intervient dans la contrainte : $st_i - st_j \leq \underline{t_{ij}}$, les contraintes de coupes 3.16 de la section 3.3.2 impliquent la succession des trains $st_i - st_j \leq -m_s^*$, si $\underline{t_{ij}} \geq m_s^*$, alors $\underline{t_{ij}}$ n'influera pas sur le critère. Ainsi le choix d'une plus petite valeur pour $\underline{t_{ij}}$ conduira à diminuer l'intervalle de succession des trains i et j.
- $\overline{t_{ij}}$ intervient dans la contrainte : $st_i - st_j \geq \overline{t_{ij}}$. Ainsi le choix d'une plus grande valeur pour $\overline{t_{ij}}$ conduira à diminuer l'intervalle de succession des trains i et j.

En définitive avec l'heuristique retenue, lors de l'affectation de $\underline{t_{ij}}$ le choix se portera pour des valeurs les plus proches de 0 et pour $\overline{t_{ij}}$ le choix se portera pour des valeurs les plus éloignées de 0. Cette heuristique tend à réduire les temps de successions des trains ($st_i - st_j$ ou δ_{ij} les plus faibles).

Ordre sur les valeurs des parcours

Avec chaque parcours pris séparément, aucune de ses caractéristiques ne permet d'effectuer un choix pour l'affecter à une variable r. Pour définir une heuristique de choix, nous avons dû calculer une caractéristique plus globale qui évalue le niveau d'incompatibilités des parcours.

Cette fonction d'évaluation a été calculée de deux façons. La première, qualifiée de *statique*, considère tous les parcours de tous les trains. La seconde, qualifiée de *dynamique*, considère les parcours des trains situés dans le voisinage du train à affecter. De plus, afin d'affiner la seconde méthode, un *profil d'incompatibilité* du voisinage du train permet de pondérer le calcul des incompatibilités.

Fonction d'évaluation statique Cette évaluation statique des parcours est obtenue par le cumul des durées des conflits pour chaque parcours en fonction de tous les autres, en aval et en amont. Ce qui revient à la fonction suivante :

$$\begin{aligned} eval_{statique}(a) &= \sum_{b \in R} (eval(a,b) + eval(b.a)), \\ avec\ eval(a,b) &= \begin{cases} -V\underline{t_{ab}} \text{ si } V\underline{t_{ab}} < 0 \\ 0 \text{ sinon} \end{cases} \end{aligned} \tag{4.1}$$

avec :

- a, b des parcours,

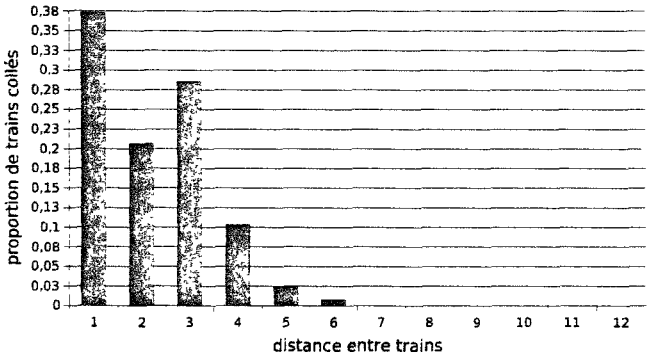

FIG. 4.1 – Distance entre les trains vérifiant les contraintes d'incompatibilités à l'égalité

- Vt_{ab} représentant la valeur prise par la variable de la borne d'incompatibilités t_{ij} lorsque le train i prend le parcours a et le train j le parcours b.

Ce type d'évaluation classe les parcours du moins conflictuel au plus conflictuel (ordre croissant des $eval_{statique}(a)$), a priori, par rapport à l'ensemble des parcours.

Profil d'incompatibilité L'évaluation du niveau d'incompatibillité d'un parcours peut aussi prendre en considération les parcours des trains situés dans le voisinage. Les trains les plus éloignés auront une probabilité plus réduite, voire nulle, d'incompatibilité. C'est d'ailleurs à partir de cette constatation que l'on a supprimé des contraintes (cf. section : 3.3.1).

Pour deux trains d'indice i et j, la fréquence des trains « collés » est un indicateur de l'incompatibilité potentielle entre les valeurs des variables parcours r_i et r_j. L'expression trains « collés » signifie que ces trains vérifient les contraintes d'incompatibilités à l'égalité (ie : $st_i - st_j = t_{ij}$).

Pour calculer un profil moyen d'incompatibilité d'un voisinage, nous avons procédé à la résolution d'un jeu d'instances de problèmes. Le jeu d'instances se base sur les 18 types d'instances réelles et aléatoires détaillées en section 3.5. En prenant des tailles de 8, 9, 10, 11 et 12 trains (des tailles plus importantes ne sont pas résolues en moins d'une heure), nous avons obtenu un total de 80 instances de problèmes. Le diagramme de la figure 4.1 montre les résultats de la moyenne des incompatibilités sur les 80 instances que le tableau 4.1 détaille.

Dans l'hypothèse où notre jeu d'instances est représentatif, on peut déduire que deux trains successifs (distance de 1) ont une chance de satisfaire la contrainte d'incompatibilité à l'égalité dans 38% des cas, les trains distants de plus de 7 trains ne semblent, quant à eux, jamais collés.

Fonction d'évaluation dynamique La fonction d'évaluation dynamique des parcours d'un train tient compte des affectations des trains situés dans *son voisinage.*

Pour un train d'indice i, le voisinage considéré dans notre fonction d'évaluation (équation 4.2) comprend les 4 trains précédant et les 4 trains suivant ce train i. En effet le diagramme de la figure

distance	1	2	3	4	5	6	7
aléa1	39%	22%	30%	12%	3%	0%	0%
aléa2	38%	21%	28%	10%	2%	1%	0%
aléa3	40%	19%	28%	15%	2%	1%	0%
TGV-MA	38%	20%	29%	10%	3%	0%	0%
CL-MA	37%	21%	30%	12%	3%	2%	0%
TGV-CL-MA	38%	20%	29%	10%	3%	0%	0%

TAB. 4.1 – Distance entre les trains vérifiant les contraintes d'incompatibilités (par type d'instance)

4.1 montre qu'à une distance supérieure à 4, les trains ont peu de chances d'avoir des contraintes d'incompatibilités satisfaites à l'égalité (donc d'être directement en conflit).

Cette fonction évalue un parcours du domaine de la variable en cours d'instanciation en fonction des parcours des trains déjà instanciés. Les trains ayant déjà leurs parcours affectés sont ceux dont la variable r_l est fixée. La fonction d'évaluation est une somme pondérée des probabilités que chaque couple de trains (i,l) soit collé. L'estimation de la probabilité que 2 trains i et l soient collés est pondérée par le paramètre ps_{l-i} du profil d'incompatibilité décrit précédemment.

$$eval_{dynamique}(a) = \sum_{l=max(0,i-4)}^{i-1} ps_{l-i} \times \underline{Vt_{r_l,a}} + \sum_{l=i+1}^{min(N,i+4)} ps_{l-i} \times \underline{Vt_{a,r_l}} \tag{4.2}$$

$$avec\ \underline{Vt_{a,r_l}} = \begin{cases} -\underline{Vt_{r_l,a}} \text{ si } \overline{Vt_{r_l,a}} < 0 \text{ et } r_l \textit{ fixée} \\ 0 \text{ sinon} \end{cases}$$

$$et\ ps_1 = 0,38.\ ps_2 = 0.2,\ ps_3 = 0,28.\ ps_4 = 0,1.$$

Dans les cas où peu de trains sont affectés autour du train i (notamment au début de l'arbre de recherche), la fonction d'évaluation dynamique définie précédemment n'est pas assez fiable. Il convient donc de définir un seuil au delà duquel cette fonction pourra être utilisée en complément de la fonction $eval_{statique}$. Ainsi la fonction d'évaluation $eval_r$ retenue regroupe la fonction d'évaluation statique et dynamique :

- Si moins de 2 variables parcours sont instanciées à une distance inférieure à 4 par rapport au train i, la fonction d'évaluation des parcours sera $eval_{statique}$ (4.1).
- Si plus de 2 variables parcours sont instanciées à une distance inférieure à 4 par rapport au train i, la fonction d'évaluation des parcours sera $eval_{dynamique}$ (4.2).

Le choix du parcours à instancier est le parcous ayant la plus petite évaluation pour $eval_r$.

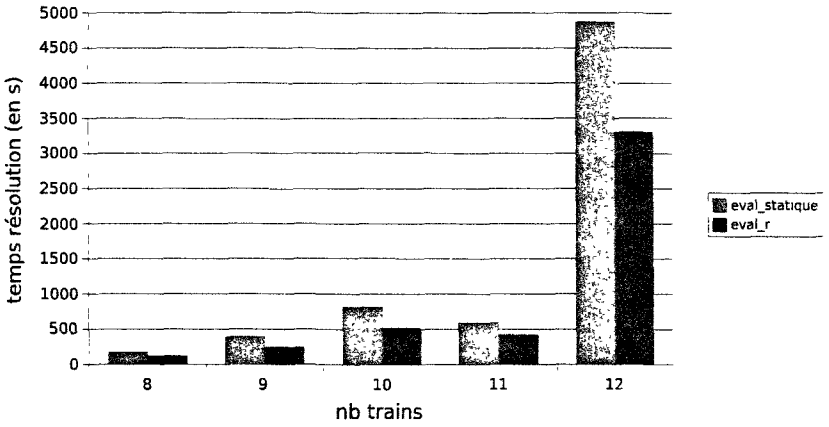

FIG. 4.2 – Impact de l'heuristique de choix de valeur sur le temps de résolution

Expérimentations Des tests ont été effectués sur un ensemble d'instances afin de comparer l'utilisation de la fonction d'évaluation $eval_{statique}$ et de la fonction d'évaluation mixte $eval_r$.

La méthode utilisée est une résolution exacte présentée en section suivante 4.3.1. Les figures 4.2 et 4.3 présentent la comparaison entre les résultats obtenus avec $eval_{statique}$ (équation 4.1) et $eval_r$.

Ces tests ont été effectués sur tous les types d'instances ferroviaires et aléatoires, présentées en section 3.5, le nombre de trains des instances traitées est inférieur à 12. En effet, au-delà le temps de résolution devient trop important (plus de 3 heures). Ces diagrammes montrent que le choix dynamique de valeurs permet d'obtenir la solution optimale et de prouver son optimalité plus rapidement. Cette heuristique dynamique permet de réduire en moyenne de 30% le temps d'obtention de la solution optimale en ce qui concerne les instances réelles et de 15% sur les instances aléatoires.

La différence entre les instances aléatoires et réelles peut s'expliquer car les instances aléatoires possèdent un petit ensemble de parcours très efficaces : les autres ne sont alors pas ou peu testés.

L'utilisation d'un choix aléatoire de parcours a impliqué une résolution en moyenne 10 fois plus longue. Des tests ont également été effectués en affectant en premier les parcours ayant la moins bonne notation par la fonction d'évaluation, le temps de résolution dans ce cas s'avère exhorbitant : il faut plus de trois heures pour résoudre des instances à 8 trains, et les instances de tailles plus importantes n'ont pu être résolues dans des temps acceptables (moins de dix heures), cet ordre donne en moyenne un temps de résolution 250 fois plus long.

4.2 Paramètres des améliorations

Dans le chapitre 3, plusieurs améliorations au modèle ont été proposées. La plupart d'entre elles impliquent l'utilisation de paramètres dont les valeurs et spécificités vont être décrites ici.

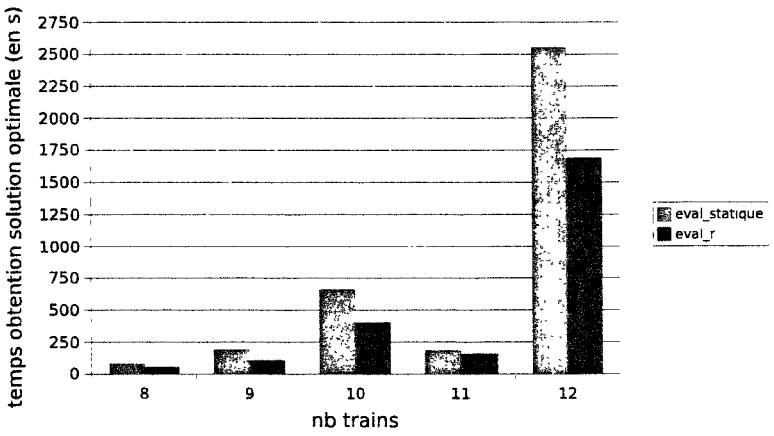

FIG. 4.3 – Impact de l'heuristique de choix de valeur sur le temps d'obtention de la valeur optimale

4.2.1 Coupes

Les coupes proposées en section 3.3.2 portent sur l'écart minimum entre les trains selon leur éloignement d'indice. Ces écarts sont obtenus pour des distances comprises entre 2 et N_s (taille maximale de la sous-séquence considérée pour l'ajout de coupes). Il s'agit alors de résoudre des problèmes de même type que l'instance considérée mais ayant moins de trains.

Ces coupes sont alors intégrées au modèle par l'intermédiaire des contraintes suivantes, où N_s correspond à la taille maximale des sous-séquences de trains considérées et m_s^* à la valeur optimale du makespan du problème à s trains :

$$m_s^* \leq st_s, \forall s = 2, \ldots, N_s \tag{4.3}$$

Ces coupes permettent d'améliorer les temps de résolution d'un problème mais pour des valeurs N_s importantes, le temps de calcul de ces coupes compense les gains obtenus. Le but des expérimentations présentées ici est de déterminer la valeur de N_s au delà de laquelle les coupes n'améliorent pas le temps global de résolution du problème.

L'histogramme 4.4 présente les temps relatifs moyens d'obtention du makespan de l'ensemble des instances réelles et aléatoires présentées section 3.5 (18 instances), pour des tailles de 10 et 11 trains. Ces tailles d'instances permettent une résolution des problèmes en des temps corrects (moins d'une heure). Les valeurs de N_s vont de 6 à 9, tel que $N_s < N$ et sont indiquées en abscisse. L'axe des ordonnées indique le temps de calcul complet normalisé par rapport au temps de calcul avec $N_s = 6$ (la moyenne des temps de résolution pour $N_s = 6$ vaut 1) sur les 18 instances considérées et pour $N = 10$ et 11. Le temps de calcul (TC), indiqué par les barres, tient compte de la résolution TR (en gris clair) du problème à N trains et du temps TS (en gris foncé) d'obtention des valeurs de makespan pour les coupes, d'où $TC = TR + TS$. Nous avons choisi de présenter la moyenne sur l'ensemble des instances car l'action de N_s est identique pour chaque instance.

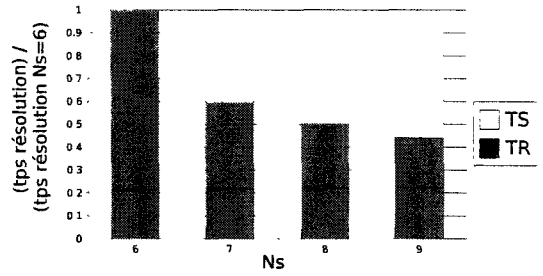

FIG. 4.4 – Impacts des valeurs de N_s sur la résolution

À partir des résultats obtenus, il s'avère que le meilleur compromis entre le temps de calcul des coupes et la réduction du temps de résolution d'un problème se situe pour la valeur $N_s = 7$: au niveau du creux du diagramme. Nous prendrons donc, pour nos résolutions futures, la valeur 7 pour N_s.

4.2.2 Borne supérieure

La borne supérieure (cf. section 3.3.2) choisie pour notre problème correspond à la solution d'un problème P' plus contraint que notre problème de capacité ferroviaire P et tel que $sol(P') \subseteq sol(P)$ où $sol(P)$ regroupe l'ensemble des solutions du problème P. On rappelle qu'une solution de ce problème P' correspond à une suite de trains empruntant la même séquence de parcours.

La contrainte ajoutée au modèle est alors :

$$st_N \leq st\_seq^*_{n_{seq}+1} \times \lfloor N/n_{seq} \rfloor + st\_seq^*_{(N \bmod n_{seq})},$$

avec $st\_seq^*_0 = 0$.

Les expérimentations qui vont être présentées ont pour but de déterminer la meilleure taille de séquence de parcours n_{seq}.

Le premier critère de choix de cette taille est le temps de résolution qui doit être inférieur à 5 minutes (moins de 10% du temps accordé pour la résolution), le second est la qualité de la solution. Le tableau 4.2 présente les temps de résolution nécessaires à l'obtention de cette borne sur l'ensemble des instances pour des tailles de n_{seq} de 6 et 7 , la valeur du makespan en étendant ces solutions sur 100 trains pour $n_{seq} = 6$ et 7 est présentée sur le tableau 4.3. La qualité de la borne est, en moyenne, meilleure pour des tailles de séquences à 7 trains qu'à 6 trains. Nous avons également testé pour des tailles de $n_{seq} = 8$, dans ce cas le temps de résolution exact dépasse le temps accordé, la solution trouvée dans le temps imparti étant bien moins bonne que celle trouvée pour $n_{seq} = 7$.

Comme le montre le tableau 4.3, en étendant la solution du problème P' à $N = 100$ trains, la qualité des solutions de cette borne supérieure est proche de celles obtenues par Delorme

type d'instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
temps de résolution ($n_{seq} = 6$)	2s	3,2s	4,6s	10s	77s	24s
temps de résolution ($n_{seq} = 7$)	29s	17s	13s	9s	123s	280s

TAB. 4.2 – Temps de résolution du problème P' permettant d'obtenir la borne supérieure

type d'instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
makespan $n_{seq} = 6$	5 920s	5 536s	5 408s	4 029s	4 046s	4 029s
makespan $n_{seq} = 7$	5 220s	5 546s	5 021s	3 872s	3 808s	3 808s

TAB. 4.3 – Makespan en étendant les solutions des problèmes P' à 100 trains

[Delorme et al.01] sur les instances réelles, avec une moyenne de 94 trains pouvant circuler sur l'infrastructure en moins d'une heure (moins de 3 600s sur le tableau) contre une moyenne de 97 pour Delorme.

4.2.3 Réduction du nombre de contraintes

En section 3.3.1, nous avons vu qu'il était possible de ne pas poser toutes les contraintes entre tous les couples de trains i et j s'ils sont séparés d'au moins N^*_{inc} indices.

Les expérimentations suivantes ont pour objectif de déterminer la valeur de N^*_{inc} pour chaque instance considérée. L'écart maximum possible entre 2 trains bt_{inc} pour chaque instance est déterminé par : $bt_{inc} = max\ |bt|,\ bt \in \underline{I} \cup \overline{I}$.

Déterminer le nombre de trains pouvant passer dans l'intervalle de temps :$[0,\ bt_{inc}]$ revient à résoudre, pour chaque type d'instance, un problème $P_{N_{inc}}$ possédant les mêmes contraintes que le problème M_0 auquel a été modifié le critère et ajouté un type de contrainte de la manière suivante :

$$\max_{N_{inc} \in [1..N]} N_{inc} \tag{4.4}$$

$$st_{N_{inc}} \leq bt_{inc} \tag{4.5}$$

La résolution d'un tel problème n'est pas faisable, en un temps raisonnable, pour les instances traitées. L'idée développée est d'utiliser les relations entre trains obtenues par les coupes (cf. section 3.3.2) comme borne supérieure de N_{inc}. En effet, les coupes sont aussi des contraintes de précédences entre les variables st_i. La propagation de ces contraintes nous permet d'obtenir des relations de précédence dont l'écart est supérieur à bt_{inc}. Parmi ces relations celle ayant le plus petit écart d'indice fournit une borne supérieure de N^*_{inc}.

Exemple Obtention de l'approximation de N_{inc}

Soit les coupes suivantes :
- $st_{i+2} \geq st_i + 5$, $\forall i \in 1, \ldots, N-2$
- $st_{i+3} \geq st_i + 15$, $\forall i \in 1, \ldots, N-3$
- $st_{i+4} \geq st_i + 25$, $\forall i \in 1, \ldots, N-4$

et soit $bt_{inc} = 45$.
Par propagation nous avons :
$st_{i+6} \geq st_i + 30$, $\forall i \in 1, \ldots, N-6$
$st_{i+7} \geq st_i + 40$, $\forall i \in 1, \ldots, N-7$
$st_{i+8} \geq st_i + 50 (\geq bt_{inc})$, $\forall i \in 1, \ldots, N-8$
La dernière relation indique qu'il ne peut y avoir plus de 9 trains dans l'intervalle [0, 50], et par conséquent dans l'intervalle [0, 45] aussi. Nous pouvons donc affecter la borne supérieure $\overline{N_{inc}} = 9$.

Le tableau 4.5 présente les valeurs des bornes $\overline{N_{inc}}$ par type d'instance (valeur moyenne pour les instances aléatoires). Ainsi, sur les instances traitées, les valeurs de $\overline{N_{inc}}$ sont comprises entre 15 et 21.

Type d'instance	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
$\overline{N_{inc}}$	15	18	21	17	20	21

TAB. 4.5 – Nombre maximum estimé de trains pouvant être en conflit par type d'instance

4.3 Résolution exacte

Afin de résoudre de manière exacte le problème de saturation d'infrastructure ferroviaire modélisé en un CSP, les algorithmes proposés par ILOG Solver [Ilog02] et par CHOCO [Choco] ont été utilisés. Il s'agit de deux ensembles de bibliothèques permettant une modélisation et une résolution de problèmes de satisfaction de contraintes. Ces bibliothèques sont utilisées ici pour un problème d'optimisation (avec critère) et non de satisfaction de contraintes. L'algorithme par défaut employé par ces bibliothèques est basé sur un branch & bound en profondeur d'abord, sur lequel des algorithmes de propagation agissent, de manière plus ou moins intensive, à chaque nouvel événement (réduction de domaine, affectation de valeur). Sur les 2 programmes implémentés, l'un en CHOCO, l'autre en Solver, la même modélisation, les mêmes contraintes, heuristiques ont été implémentées. Chacune de ces bibliothèques possède ses particularités, avantages et inconvénients pour le problème traité. Ils sont décrits ci-dessous.

4.3.1 Ilog-Solver

Le programme utilisant les bibliothèques d'ILOG-Solver [Ilog02] et ILOG-Concert [Ilog01] est écrit en C++, la version de Solver est la 5.3, celle de Concert la 1.2. Les logiciels et bibliothèques proposés par Ilog font référence dans le domaine de la recherche opérationnelle, grâce à CPLEX. Il en va de même pour Solver et Concert. Ce sont des produits puissants et très performants. Ils possèdent néanmoins deux inconvénients :

- le coût élevé des licences,
- la "boîte noire" des algorithmes implémentés, il n'est pas possible d'avoir un contrôle parfait des traitements réalisés lors de la recherche.

4.3.2 CHOCO

même s'améliore

L'implémentation de l'algorithme de résolution avec CHOCO a été réalisée en collaboration avec Hadrien Cambazard de l'École des Mines de Nantes. Cette implémentation a été réalisée en JAVA et avec la version 0.9 de CHOCO. L'intérêt de l'utilisation de ces bibliothèques est le caractère libre de son code source, ce qui implique qu'il est possible de contrôler, de modifier et de savoir exactement toutes les opérations réalisées par l'algorithme. Les inconvénients de ces bibliothèques sont :

- la documentation n'est pas très détaille, même si elle s'améliore depuis les versions 1.0 notamment grâce au site web,
- certaines contraintes spécifiques ne sont pas implémentées ou optimisées (pour la version 0.9), comme par exemple l'utilisation des « table-constraint » (contrainte du même type que notre contrainte d'énumération 3.4) qui était en cours de développement lors de notre étude et donc non encore optimisée.

4.3.3 Comparaison des 2 bibliothèques

Le tableau 4.6 rend compte des temps nécessaires pour la résolution des différentes types d'instances présentées en section 3.5. Ces résultats portent sur des tailles d'instances comprises entre 6 et 9 trains.

instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
temps $\frac{CHOCO-SOLVER}{SOLVER}$	65%	71%	68%	65%	66%	67%

TAB. 4.6 – Comparaison des temps de résolution par les bibliothèques CHOCO et ILOG SOLVER

Comme nous pouvons le remarquer, CHOCO n'est pas en mesure de rivaliser avec ILOG Sol-

ver sur notre problème, le temps de résolution est 2,5 fois plus long que pour SOLVER et plus le nombre de trains à traiter est important et plus la différence croît. La faiblesse relative des performances de CHOCO par rapport à Ilog-Solver peut s'expliquer par :

- les temps d'exécution des applications développées en Java sont souvent moins performants que celles réalisés en C++,
- l'implémentation non optimale (en cours de développement au moment de notre étude) des table-constraint (contraintes 3.4 permettant l'énumération tuples admissibles pour les variables $r_i, r_j, \underline{t_{ij}}, \overline{t_{ij}}$).

4.4 Résolution approchée

Intéressons-nous à présent à la résolution approchée du problème ferroviaire. La majorité des métaheuristiques existantes en programmation par contraintes porte sur la recherche locale (cf. section 2.7) et sur la recherche à arborescence tronquée (cf. section 2.6). Notre choix s'est porté sur 3 types d'algorithmes :

- la recherche avec arborescence tronquée Depth-bounded Discrepancy Search (DDS, cf. section 2.6.2),
- la recherche à voisinage large LNS (Large Neighborhood Search , cf. section 2.7.2),
- la recherche à voisinage variable VNS (Variable Neighborhood Search : cf. section 2.7.3).

Outre l'étude propre de ces méthodes, nous nous sommes intéressés à leur combinaison. Par ailleurs, la métaheuristique GRASP implémentée et présentée par Delorme [Delorme03] a également été utilisée et quelque peu modifiée afin de fournir une solution initiale à notre algorithme.

La suite de cette section présente la manière dont ces algorithmes ont été utilisés pour résoudre le problème ferroviaire. Leurs paramètres sont ainsi décrits et leurs valeurs données.

4.4.1 Depth-bounded Discrepancy Search (DDS)

Plusieurs méthodes de recherche arborescente tronquée ont vu le jour depuis quelques années (cf. section 2.6). Ces méthodes sont basées sur le concept de divergence afin d'améliorer le processus de recherche. Une divergence correspond à ne pas suivre la première valeur préconisée par l'heuristique de choix de valeurs. Ces techniques peuvent être utilisées de deux manières :

- pour une résolution exacte sans limite sur le nombre de divergences,
- pour une résolution approchée en limitant le nombre de divergences.

L'heuristique de choix de valeur de la section 4.1.2 est de bonne qualité. Il devient donc envisageable de l'utiliser avec une méthode de recherche à arborescence tronquée pour une résolution approchée afin de limiter le temps de résolution.

L'algorithme Depth-bounded Discrepancy Search (cf. section 4.4.1), proposé par Walsh [Walsh97], a été choisi. Par la suite, le principe de cet algorithme et de ces paramètres sont rappelés.

Principe

L'algorithme Depth-bounded Discrepancy Search (DDS) est inspiré de l'algorithme Limited Discrepancy Search (LDS) (cf. section 2.6.1). Par rapport à LDS, l'algorithme DDS considère en plus le fait que l'heuristique de choix de valeurs risque de faire plus d'erreurs au début de l'arbre qu'à la fin. Or, l'heuristique dynamique proposée en section 4.1.2 tient compte des affectations précédemment effectuées pour choisir la valeur à affecter. Elle devrait donc accroître ses performances au fur et à mesure de l'affectation des variables, ce qui constitue le principe de la stratégie DDS. On peut donc s'attendre à ce que l'algorithme DDS soit performant avec cette heuristique d'ordre. La section suivante présente les paramètres et les valeurs testées pour l'algorithme DDS utilisé.

Paramétrage

L'algorithme DDS implémenté par ILOG est contrôlé par 3 paramètres :

- $depth$: les nœuds avec une profondeur inférieure à $depth$ seront explorés en premier, puis les nœuds allant jusqu'à $2 \times depth$, puis jusqu'à $3 \times depth$, ... Ainsi ce paramètre affectera l'ordre d'exploration de l'arbre de recherche, par exemple choisir une très grande valeur pour $depth$ reviendra à effectuer une recherche en profondeur d'abord.
- $width$: les chemins de l'arbre de recherche ayant moins de $width$ anomalies (discrepancy) seront explorés en premier.
- $disc$: Les nœuds présentant plus de $disc$ anomalies ne seront pas explorés.

Les valeurs précisées ci-dessous sont celles utilisées pour la résolution du problème n'utilisant que l'algorithme DDS, à savoir :

- $depth$: Aux vues des tailles d'instances que nous pouvons traiter (maximum 14 trains) nous avons limité $depth$ à prendre les valeurs : 4, 6 ou 8. Des valeurs plus petites donnent la même qualité de solutions avec un temps de résolution supérieur, et des valeurs plus grandes donnent une qualité trop médiocre.
- $width$: Après plusieurs essais, il s'est avéré qu'un nombre de divergences inférieur à 8 donne de bonnes solutions, aussi nous avons choisi comme valeurs à tester : 6, 8 et 10. À noter que la documentation d'ILOG SOLVER sur ce paramètre n'est pas correcte, cette erreur a été confirmée par les services d'ILOG et devrait maintenant être corrigée.
- $disc$: les nœuds présentant plus de $disc$ anomalies ne seront pas explorés. Comme dit précédemment les tests effectués ont montré que la solution optimale déviait rarement de plus de 8 anomalies par rapport au choix heuristique. Les valeurs considérées pour les tests sont donc : 8, 10 et 12.

Les résultats sur les différentes combinaisons de valeurs de paramètres sont détaillés en section 5.2.1.

4.4.2 Recherche à voisinage Large (LNS)

Comme indiqué en section 2.7.2, la métaheuristique de recherche à voisinage Large (Large Neighborhood Search : LNS) est une méthode par relaxation et ré-optimisation successive. Son utilisation requiert de définir :

- une solution initiale,
- un type de voisinage,
- une méthode de résolution,
- un critère d'arrêt.

La figure 4.5 propose un résumé du schéma LNS implémenté, dont les différentes phases et paramètres sont expliqués ci-après. Les deux principales différences par rapport à l'algorithme 2.7.1 portent sur le fait que :

- l'ensemble des variables est relâché une fois par voisinage (sauf si le critère d'arrêt est levé),
- la taille du voisinage évolue au cours de la recherche.

Les notations utilisées sur ce schéma sont :

- $Neigh_{size}$: taille du voisinage considéré,
- Δ_{Neigh} : pas d'augmentation de la taille du voisinage,
- TR : temps donné à LNS (critère d'arrêt).

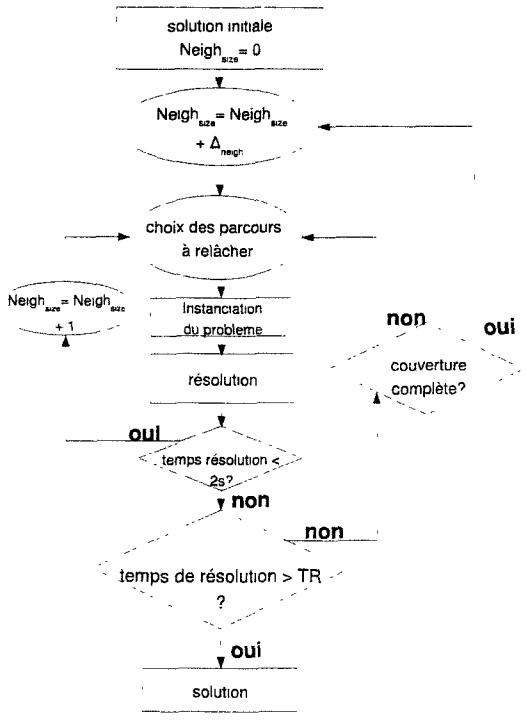

FIG. 4.5 – Schéma de l'algorithme LNS

Solution initiale :

La solution initiale est un paramètre essentiel des méthodes de recherche approchées. Dans notre étude, trois types de solutions initiales ont été utilisés :

- Une solution initiale de mauvaise qualité : elle est composée de trains empruntant tous le parcours le moins bien classé par la fonction d'évaluation $eval_{statique}$ vue en section 4.1. Ce type de solution initiale permet d'évaluer la sensibilité des performances de l'algorithme vis à vis de la solution initiale.
- La solution obtenue par la résolution du problème (P') (cf. section 3.3.2) pour le calcul de la borne supérieure. Le problème (P') correspond au problème initial (P) auquel des contraintes obligeant les trains à prendre des séquences de parcours identiques, ont été ajoutées. Il s'agit d'une solution réalisable et de bonne qualité.
- La solution trouvée par l'algorithme GRASP, proposé par Delorme [Delorme03]. Cette méthode résoud le même problème ferroviaire, si ce n'est qu'il est modélisé en un problème de Set Packing (cf. section 1.4.1 et 1.4.4). La métaheuristique a été proposée initialement par Féo et Resende [Fo et al.89]. C'est une méthode multi-départ en deux phases. La première phase correspond à la construction d'une solution initiale via une procédure gloutonne aléatoire, la seconde correspond à une recherche locale sur cette solution initiale. Plusieurs composants ont été apportés (cf. Pitsoulis et Resende [Pitsoulis et al.02] pour une bibliographie complète) afin de contrôler l'aspect aléatoire de la phase initiale, l'utilisation de techniques à base de mémoire et d'apprentissage, le couplage avec d'autres méthodes [Resende et al.02]. Pour un problème donné cinq éléments caractérisent son implémentation :
  - Trois types d'algorithme glouton : construction, destruction de solution et construction avec apprentissage sur les variables déjà fixées à 1 dans d'autres solutions, afin de diversifier les solutions initiales.
  - L'importance de la composante aléatoire de la phase gloutonne : contrôlée par un paramètre α.
  - Le type de recherche locale et de voisinage utilisé : $0-1, 1-1, 2-1, 1-2$ échanges.
  - La phase d'intensification : algorithme de path-relinking ou de recomposition de chemins (passage d'une solution à une autre avec pour origine et destination des solutions obtenues précédemments) proposé par Glover et Laguna [Glover et al.97], c'est une méthode visant à générer le chemin reliant deux solutions de bonne qualité.
  - Le critère d'arrêt : fixé à 15 minutes au lieu des 30 initialement allouée, afin de laisser 15 minutes à la métaheuristique implémentée.

  Les problèmes traités par GRASP sont des modèles de type SPP « Set Packing Problem » (dont une description est donnée par Gondran et Minoux [Gondran et al.95]), problèmes à variables binaires et contraintes de disjonction, ils ont les mêmes caractéristiques que le modèle CSP. Une table de correspondance entre variables a été faite et permet de passer rapidement d'un modèle à l'autre (moins de 30 secondes).

Le critère du modèle SPP traité par GRASP (maximiser le nombre de trains) et celui de notre modèle CSP (minimiser le makespam) n'étant pas identiques, nous avons utilisé une méthode de compensation entre les deux types de solutions :

- si la solution de GRASP comporte plus de N trains (N est le nombre de trains du modèle CSP), les N premiers de la solution sont gardés,
- si la solution de GRASP comporte moins de N trains, elle est complétée en ajoutant des trains dont le parcours est celui défini par l'heuristique $eval_{dynamique}$ (équation 4.2).

Voisinage :

La quantification du voisinage porte sur les trains et plus précisément, elle est basée sur les variables parcours (r_i). Ainsi un certain ensemble de variables r_i est relâché, notons F l'ensemble des indices des variables r_i relâchées.

L'ensemble des variables des bornes d'incompatibilités ($\underline{t_{ij}}$, $\overline{t_{ij}}$) comportant au moins une contrainte commune avec les r_i relâchées le sont à leur tour. En d'autres termes, cela correspond à l'ensemble des $\underline{t_{ij}}$, $\overline{t_{ij}}$, tel que $i \in F$ ou $j \in F$.

Enfin, toutes les variables dates de départs (st_i), et toutes les variables d'écart (δ_{ij}) sont relâchées. En effet, les dates de départ des trains sont influencées par les variables des bornes d'incompatibilités et toutes les autres dates de départs, ainsi, changer la valeur d'une seule date de départ peut influer sur toutes les autres. Une fois l'ensemble des parcours et bornes d'incompatibilités fixées, le temps nécessaire à l'affectation des variables dates de départs est négligeable grâce aux coupes ajoutées 3.16.

Exemple de voisinage

Soit une instance quelconque à 6 trains et une taille de voisinage de 1, il y a donc pour ce problème 6 variables :

- variables parcours : $r_1, r_2, r_3, r_4, r_5, r_6$,
- variables bornes d'incompatibilités : $\underline{t_{12}}$, $\overline{t_{12}}$, $\underline{t_{13}}$, $\overline{t_{13}}$, $\underline{t_{14}}$, $\overline{t_{14}}$, $\underline{t_{15}}$, $\overline{t_{15}}$, $\underline{t_{16}}$, $\overline{t_{16}}$, $\underline{t_{23}}$, $\overline{t_{23}}$, $\underline{t_{24}}$, $\overline{t_{24}}$, $\underline{t_{25}}$, $\overline{t_{25}}$, $\underline{t_{26}}$, $\overline{t_{26}}$, $\underline{t_{34}}$, $\overline{t_{34}}$, $\underline{t_{35}}$, $\overline{t_{35}}$, $\underline{t_{36}}$, $\overline{t_{36}}$, $\underline{t_{45}}$, $\overline{t_{45}}$, $\underline{t_{46}}$, $\overline{t_{46}}$, $\underline{t_{56}}$, $\overline{t_{56}}$,
- variables dates de départs : $st_1, st_2, st_3, st_4, st_5, st_6$,
- variables d'écart : $\delta_{12}, \delta_{13}, \delta_{14}, \delta_{15}, \delta_{16}, \delta_{23}, \delta_{24}, \delta_{25}, \delta_{26}, \delta_{34}, \delta_{35}, \delta_{36}, \delta_{45}, \delta_{46}, \delta_{56}$.

Ainsi si la variable de base relâchée est la variable r_3, alors il faudra également relâchées les variables :

- $\underline{t_{13}}$, $\overline{t_{13}}$, $\underline{t_{23}}$, $\overline{t_{23}}$, $\underline{t_{34}}$, $\overline{t_{34}}$, $\underline{t_{35}}$, $\overline{t_{35}}$, $\underline{t_{36}}$, $\overline{t_{36}}$,
- toutes les varaibles dates de départs,
- toutes les variables d'écart.

L'implémentation de cette méthode a fait apparaitre deux anomalies que l'assistance technique d'Ilog n'a pas su résoudre :

- La fonction permettant de relâcher une variable ne libère pas la mémoire allouée précédemment (ce qui implique une fin de résolution à plus de 1,5 Go contre 10 Mo en début de recherche)
- Les contraintes de type *enum* restent actives, même après leur libération.

Pour pallier à ces problèmes nous avons implémenté une méthode de pseudo libération revenant à instancier pour chaque résolution un nouveau problème, soit pour chaque ré-optimisation le processus suivant :

1. réinstancier tout le problème : variables, contraintes,
2. affecter aux variables fixées les valeurs de la solution de l'itération précédente,
3. résoudre l'instance du problème,
4. sauvegarder la solution.

La mise en place de cette méthode requiert moins de 0,5 seconde par ré-optimisation et en moyenne, moins de 0,5% du temps de résolution. Cependant, cette réinstanciation a dû être prise en compte dans notre schéma de l'algorithme LNS.

Le nombre de variables parcours relâchées, dénoté $Neigh_{size}$, débute à 1 et est augmenté au fur et à mesure de la résolution. De trop petites tailles de voisinage peuvent être ré-optimisées très rapidement, ce qui entraîne un temps de résolution bien inférieur au temps de chargement des données (dû à notre réinstanciation complète du problème à chaque ré-optimisation). Afin de réduire la part du temps de traitement prise par ce chargement, une première boucle interne (cf. figure 4.5) impose d'augmenter la taille du voisinage tant que le temps de ré-optimisation est inférieur à 2 secondes. Une fois la taille minimale de $Neigh_{size}$ établie, un test de courverture de l'ensemble des variables parcours impose que chaque variable ait été au moins une fois dans le voisinage. Enfin, tant que le test sur la limite de traitement n'est pas satisfait, des ré-optimisations avec des voisinages toujours plus grands sont réalisées. L'augmentation de la taille du voisinage se fait par un pas de 1 (respectivement 2) pour la version nommée Δ_1 (respectivement Δ_2).

Shaw [Shaw98] précise qu'un choix totalement aléatoire du voisinage n'est pas une bonne stratégie, d'autant plus que dans notre problème les trains proches ont plus de contraintes satisfaites à l'égalité que les trains éloignés. Ainsi la modification des valeurs de variables d'un train i aura un impact plus important sur le train $i+1$ que sur le train $i+K$ (K grand). Relâcher les variables parcours du train i et $i+1$ sera alors plus intéressant que relâcher celles des trains i et $i+K$. Par conséquent, les variables choisies pour un voisinage forment des séquences de trains $(i, i+1, \ldots)$. Pour le changement de voisinage, nous avons testé deux approches :

1. Chronologique : les voisinages sont constitués de variables parcours correspondantes aux séquences de trains qui se recouvrent successivement de Rec trains. La formulation de ces

séquences est :

$$\min(1+(i-1)\times(Neigh_{size}-Rec), N-Neigh_{size}+1), \ldots,$$
$$\min(Neigh_{size}+(i-1)\times(Neigh_{size}-Rec), N)$$

avec $i = 1, \ldots, \lfloor \frac{N}{(Neigh_{size}-Rec)} \rfloor$. Le paramètre Rec de recouvrement des séquences a été calculé pour obtenir au plus près un taux de 40% : $Rec = \lceil Neigh_{size} \times \frac{40}{100} \rceil$. Le schéma de la figure 4.6 présente un exemple de changement de voisinage dans l'ordre chronologique pour une instance de 14 trains avec $Neigh_{size} = 3$ et $Rec = 1$.

2. Aléatoire : les voisinages sont aussi constitués des variables parcours correspondant aux séquences de trains qui globalement se recouvrent de Rec trains. La formulation des séquences est donc identique, mais les indices i sont choisis aléatoirement dans l'intervalle $[1, \ldots, \lfloor \frac{N}{(Neigh_{size}-Rec)} \rfloor]$ en évitant les doublons. Le schéma de la figure 4.7 présente un exemple de changement de voisinage dans un ordre aléatoire pour une instance de 14 trains avec $Neigh_{size} = 3$ et $Rec = 1$

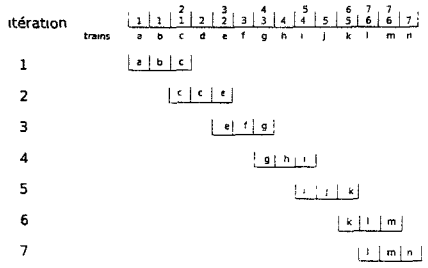

FIG. 4.6 – Exemple de changement chronologique de voisinage

Algorithme pour la ré-optimisation

Deux algorithmes de résolution ont été utilisés pour résoudre les sous-problèmes générés par les relaxations :

- L'algorithme de résolution exacte d'Ilog-Solver présenté en section 4.3.1.
- L'algorithme DDS présenté en section 4.4.1. Dans Shaw [Shaw98], De Givry et Jeannin [De givry et al.06] et Rousseau et al. [Rousseau et al.99], un algorithme de type arborescence tronquée semble être un bon candidat pour la stratégie LNS car il peut mettre à profit les variables déjà affectées. L'algorithme DDS est donc tout à fait approprié pour les phases

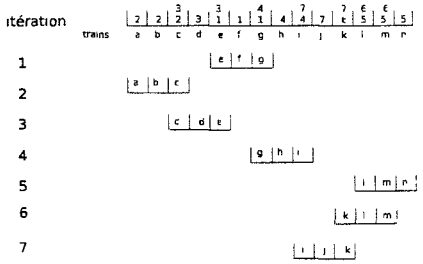

FIG. 4.7 – Exemple de changement aléatoire de voisinage

de ré-optimisation de LNS. En effet les sous-problèmes à résoudre sont plus contraints que le problème maître (toutes les variables libres : avec leur domaine de valeurs initiales). Ainsi les heuristiques de choix de valeurs commettent moins d'erreurs ce qui se traduit par peu de divergences par rapport au choix heuristique. De plus, l'heuristique de choix de valeur dynamique implémentée (cf. équation 4.2) sera plus performante, son efficacité étant directement liée au nombre de variables parcours déjà fixées.

Critère d'arrêt

Le critère d'arrêt choisi est le temps de résolution. Il a été établi à 15 minutes quelque soit le type d'algorithme employé pour la ré-optimisation. Ce temps alloué à la résolution est noté TR.

La figure 4.5 montre que plusieurs tailles de voisinage peuvent être testées. Ce changement de la taille du voisinage vise à sortir des optima locaux et ainsi introduire une pseudo diversification lors de la recherche (cf. Shaw [Shaw98]).

Dans l'implémentation de l'algorithme LNS présenté, la taille du voisinage est incrémentée lorsqu'une boucle de ré-optimisation a couvert l'ensemble des trains. Ce critère de couverture ne garantit en rien qu'avec la taille de voisinage courante, il ne soit pas encore possible de trouver une meilleure solution. Nous nous sommes donc intéressés à une méthode qui guide le changement de la taille du voisinage sur un critère moins arbitraire. Pour cela, une version de l'algorithme de recherche à voisinage variable (Variable Neighborhood Search) a été implémentée.

4.4.3 Recherche à voisinage variable (VNS)

La recherche à voisinage variable (Variable neighborhood search : VNS, cf. section 2.7.3) proposée par Mladenovic et Hansen [Mladenovic et al.97] est une métaheuristique dont le principe de base est de garder le même voisinage tant que la solution est améliorée, puis de changer de voisinage.

En suivant ce principe, cette métaheuristique a été adaptée à notre problème et ajoutée au schéma LNS de la manière suivante :

- Les voisinages considérés sont ceux présentés dans la partie précédente, basés sur les variables parcours. Les différents voisinages utilisés se distinguent par le nombre de variables parcours relâchées.
- Garder la même taille de voisinage tant que des améliorations sont apportées. En effet, dans le cas où une amélioration a lieu à l'itération it, il se peut que pour un ensemble de variables relâchées à une itération précédente $it - l, l = 1, \ldots, it - 1$ (avec le même voisinage), cette nouvelle solution permette une amélioration sur cet ensemble $(it - l)$.
- Augmenter la taille du voisinage de Δ_{Neigh} unités si aucune amélioration n'a été constatée.

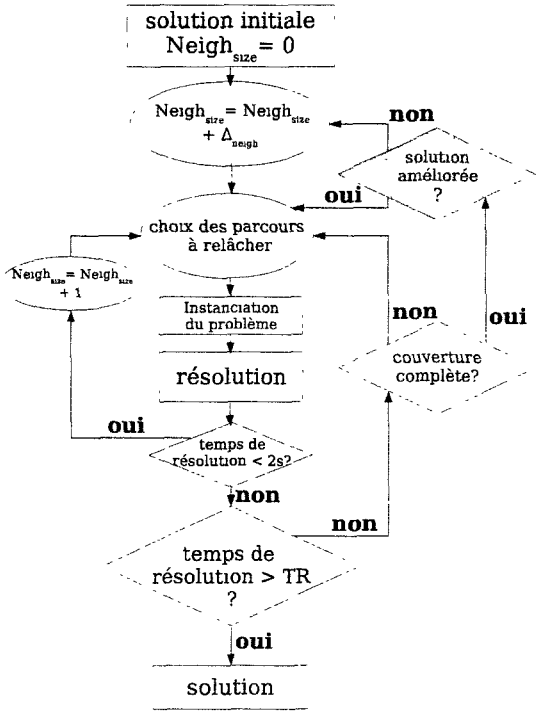

FIG. 4.8 – Schéma complet de l'algorithme LNS+VNS

Le schéma complet de l'algorithme, nommé LNS+VNS, est présenté à la figure 4.8. Il est similaire à la métaheuristique VND (Variable Neighborhood Descent) proposé par Hansen et Mladenovic dans [Hansen et al.01]. La principale différence porte sur le fait que les tailles de voisinages

ne reviennent jamais à leur valeur initiale, mais sont toujours augmentées. En effet, l'optimum local pour un voisinage donné est inclus dans l'optimum local d'un voisinage plus grand (modulo le recouvrement de variables choisis décrit dans cette section par *Rec*). L'intérêt de commencer avec de petites tailles de voisinages est d'améliorer rapidement la qualité de la solution (des sous-problèmes de petites tailles se résolvent beaucoup plus vite).

4.5 Conclusion

Dans ce chapitre, nous avons décrit les différents algorithmes pour la résolution du problème de saturation de l'infrastructure ferroviaire. Ces algorithmes peuvent également être utilisés pour résoudre le problème de faisabilité d'une grille horaire en fixant les variables selon les connaissances sur la grille horaire établie à tester.

La méthode de résolution exacte s'appuie sur les bibliothèques d'ILOG Solver ou sur les bibliothèques CHOCO. Plusieurs traitements ont été proposés afin d'améliorer la résolution, que ce soit par la ré-écriture de contraintes, l'ajout de coupes, l'ajout de borne supérieure ou l'élimination de contraintes issues des spécificités du problème. La méthode de résolution approchée, nommée LNS+VNS, combine les algorithmes des métaheuristiques de recherche à voisinage variable (VNS), voisinage large (LNS) et l'heuristique DDS.

Le chapitre suivant rend compte de la pertinence de ces propositions par la présentation des résultats obtenus sur chacun des algorithmes présentés.

Chapitre 5

Expérimentations numériques

Le problème, sa modélisation et les algorithmes de résolution ont été présentés, cette partie concerne les résultats permettant de valider ce travail. Le modèle utilisé pour la résolution de ce problème est celui de référence vu en section 3.2 : $M^0_{ref} = M_{12345}$. Il inclut l'ensemble des améliorations :

- contrainte $NGBS$ (amélioration 1),
- diminution du nombre de variables et contraintes (amélioration 2),
- coupes (amélioration 3),
- borne supérieure (amélioration 4),
- ré-écriture de la contrainte d'incompatibilité (amélioration 5).

Les heuristiques d'ordre, les différents algorithmes de résolution décrits en sections précédentes ont été codés en C++ et tournent sur un PC sous linux, processeur cadencé à 2.4GHz, 1Go de RAM.

Plusieurs résultats présentés dans cette section portent sur la comparaison de deux modèles ou deux méthodes de résolution. La convention de calcul du ratio de comparaison que nous avons utilisée est indiquée dans le tableau 5.1.

Comparaison	Ratio
A/B	$\frac{valeur_A - valeur_B}{valeur_B}$

TAB. 5.1 – Calcul d'un ratio de comparaison

Le critère étudié tient aussi bien compte de la qualité des solutions que du temps de traitement. Les paramètres $valeur_A$ et $valeur_B$ seront soit la valeur du critère ou le temps de résolution pour le modèle ou la méthode de résolution A, respectivement B. Une valeur négative du ratio indiquera que A est inférieure à B au regard du critère considéré.

Afin de ne pas surcharger les tableaux présentés dans cette section, les valeurs pour les instances aléatoires sont regroupées par moyenne des 5 instances ayant le même nombre de parcours. Ainsi,

l'étiquette *aléa* représente la moyenne des 5 instances de types aléa1 (en l'occurence 20 parcours). Les tableaux de la section annexes 5.3 présentent les détails pour ces types d'instances.

Une première partie est dédiée à la présentation des instances utilisées pour évaluer les différents modèles et techniques de résolution. Puis la présentation des résultats s'organise de la façon suivante :

1. les heuristiques d'ordre,
2. les différents améliorations sont évaluées par la comparaison de modèles, le modèle de référence prendra pour valeur le meilleur des 2 modèles. (cf. section 3.2). Le modèle de référence initial M^0_{ref} est le modèle comportant toutes les améliorations proposées.
3. les algorithmes de résolution approchée présentés précédemment sont évalués sur l'ensemble des instances.

5.1 Heuristiques d'ordre et améliorations

Les résultats exposés dans cette section concernent les heuristiques de choix de variables, les différentes améliorations et la résolution exacte proposées pour la modélisation et résolution du problème ferroviaire. Ils reprennent et complètent ceux présentés dans [Degoutin et al.05c] et [Degoutin et al.05a]. Ces résultats concernent en majorité des résolutions exactes sur de petites tailles d'instances (inférieures à 12 trains). Ces tailles sont du même ordre que les tailles de voisinages utilisées par les métaheuristiques.

5.1.1 Ordre d'énumération

L'objectif de cette section est de déterminer le meilleur ordre d'énumération des variables parmi les deux ordres retenus dans la 2.4.2 :

- t-r-st : énumération des variables d'incompatibilités ($\underline{t_{ij}}$, $\overline{t_{ij}}$), puis des variables dates de départ (st_i),
- r-t-st : énumération des variables parcours (r_i), puis des dates de départ (st_i).

Pour les expérimentations présentées, l'algorithme de résolution utilisé est un algorithme d'énumération complet utilisant les bibliothèques ILOG SOLVER. Étant donné le lien fort entre les variables parcours (r_i) et bornes d'incompatibilités (t_{ij}) par la contrainte d'énumération 3.4, une fois les variables parcours affectées, les variables bornes d'incompatibilités ont leur domaine très fortement restreint (réciproquement si les variables t_{ij} sont affectées en premier).

Ces deux types d'énumérations ont été testés sur l'ensemble des 18 types d'instances (de la section 3.5). La taille des instances traitées est de 8 et 9 trains. Nous n'avons pas expérimenté des instances avec des tailles plus grandes à cause du temps de résolution trop important. Le modèle utilisé pour ces résolutions est le modèle de référence, en l'occurence le modèle M_{12345}.

L'ordre d'affectation des valeurs est celui retenu en section 4.1.2, à savoir :

- tester les valeurs les plus proches de 0 pour $\underline{t_{ij}}$ et les plus éloignées de 0 pour $\overline{t_{ij}}$,
- tester les valeurs selon l'ordre donné par la fonction d'évaluation dynamique $eval_r$ définie en section 4.1.2.

Le tableau 5.2 présente les temps obtenus pour les 2 types d'ordre d'énumération, nous avons laissé tourner ces résolutions plus longtemps (jusqu'à plus de 7 heures) afin d'obtenir les ratios.

type d'instances		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
8 trains	*r-t-st*	0.44 s	0.35 s	1.5 s	0.2 s	4.2 s	287,4 s
	t-r-st	2.52 s	10.54 s	657.07 s	6.65 s	79.7 s	2183.41 s
9 trains	*r-t-st*	0.94 s	1.29 s	3.8 s	23.71 s	21.3 s	588.33 s
	t-r-st	18.7 s	562.31 s	1736.1 s	10 062.36 s	9290.26 s	23 251.4 s

TAB. 5.2 – Comparaison des ordres de choix de variables

Ces résultats indiquent que l'affectation des parcours en premier est bien plus efficace que l'affectation des bornes d'incompatibilités. La résolution en affectant les parcours en premier est en moyenne 25 fois plus rapide pour l'ensemble des instances. En effet, la variable parcours r est centrale dans la structure du problème : l'instanciation de r_i implique une propagation importante. Une autre explication est l'efficacité de l'heuristique de choix de valeurs associée aux parcours (cf. heuristique $eval_r$ 4.2) comparée au choix de valeur utilisé pour l'heuristique sur les bornes d'incompatibilités (cf. section 4.1.2).

Le facteur multiplicatif engendré par l'augmentation de la taille de l'instance d'un train est de 99 pour l'énumération des variables bornes d'incompatibilités en premier en moyenne contre 9 si les variables parcours sont énumérées d'abord.

Suite à ces résultats, l'heuristique de choix de variables retenue est r-t-st utilisant l'heuristique $eval_r$.

5.1.2 Impacts des améliorations

Cette partie présente l'impact des différentes améliorations apportées à la modélisation. Pour réaliser cette évaluation, les expérimentations ont été effectuées sur des résolutions exactes, et sur des tailles d'instances inférieures à 12 trains en raison du temps de calcul nécessaire (moins d'une heure de calcul).

Dans cette section, le modèle de référence initial est le modèle M_{12345} de l'équation 3.24, ce modèle intègre l'ensemble des améliorations vues au chapitre 3. Pour chaque amélioration testée, le modèle de référence est mis à jour par la fonction de récurrence décrite 3.25 en section 3.2. ce modèle référence prendra pour valeur le meilleur des 2 modèles testés :

- modèle de référence,
- modèle de référence sans l'amélioration testée.

NGBS

La contrainte $NGBS(r, st)$ décrite en section 3.3.4 permet d'éliminer les affectations symétriques des variables de trains dont les dates d'entrées sont identiques. Pour cela, elle utilise le concept de no-good : chaque fois que 2 trains auront une même valeur de st, la permutation des valeurs de parcours sera interdite. Le tableau 5.3 présente la comparaison du modèle de référence, en l'occurence le modèle complet M_{12345} utilisant cette contrainte $NGBS$, et du modèle M_{2345} sans cette contrainte. Ces résultats sont la moyenne des ratio obtenus pour l'ensemble des instances et des tailles d'instances allant de $N = 9, \ldots, 12$ trains. Comme pour la section précédente, deux types de ratio sont présentés :

- un ratio sur le temps de résolution : temps,
- un ratio sur le nombre de nœuds explorés pour trouver la solution optimale : # nœuds.

Le détail pour les instances à 12 trains est également proposé.

type d'instances		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
$N = 9, \ldots, 12$:	# nœuds	0%	0%	-1%	-1%	-1%	-1%
M_{2345} / M_{12345}	temps	2%	3%	0%	2%	2%	1%
# nœuds	M_{12345}	6897	6724	3780	85461	102338	3076578
($N = 12$)	M_{2345}	6897	6727	3784	86859	102345	3120074
temps ($N = 12$)	M_{12345}	2.13 s	7.33 s	16.77 s	56.9 s	134.76 s	2624.76 s
	M_{2345}	2.11 s	7.26 s	16.75 s	55.47 s	125.43 s	2305.34 s

TAB. 5.3 – Impact de la contrainte NGBS

Comme nous pouvons le voir, bien que cette contrainte permette une faible diminution du nombre de nœuds explorés lors de la résolution, le temps de résolution quant à lui augmente. Ainsi, le temps de traitement de cette contrainte est plus important que le temps passé pour explorer les nœuds à valeur symétrique.

Compte tenu de l'ordre d'instanciation des variables retenu ($r - t - st$), une contrainte sur un no-good ne déclenchera des réductions de domaines qu'à l'issue de l'instanciation de toutes les variables parcours, bornes puis des variables dates de départs st_i qui précèdent les variables st_i du no-good. Ces propagations n'interviendront donc que dans des parties basses de l'arbre de recherche ce qui explique la faible diminution du nombre de nœuds explorés.

Ainsi à la vue des résultats, pour la suite des expérimentations, nous utiliserons le modèle M_{2345}, d'où $M^1_{ref} = M_{2345}$ (modèle sans la contrainte $NGBS$).

Diminution du nombre de contraintes

Dans la section 3.3.1, nous avons vu que le temps d'occupation de l'infrastructure par un train peut être borné par une valeur b_{tinc}. Cette remarque nous permet de limiter le nombre de contraintes

d'incompatibilités 3.6 à poser. Tout couple de trains ayant un écart entre leur date d'entrée supérieure à bt_{inc} a obligatoirement ses contraintes d'incompatibilité satisfaites. Cette valeur temporelle n'est pas directement utilisable pour instancier notre modèle, il faut la traduire en écart d'indice dans la séquence des trains. Aussi nous avons cherché à déterminer le nombre maximum de trains pouvant circuler sur chaque instance testée, dans ce laps de temps. Ce nombre de trains a été nommé N^*_{inc} et une évaluation supérieure en a été déterminé grâce aux coupes (cf. section 4.2.3). Sur l'ensemble des instances générées, la valeur maximale des bornes des domaines d'incompatibilités bt_{inc} implique une valeur de N^*_{inc} comprise entre 15 et 21 suivant le type d'instance considéré, comme présenté dans le tableau 4.5 de la section 4.2.3.

L'intérêt de la suppression de ces contraintes sera montré à partir de deux expérimentations, l'une avec une méthode de résolution exacte, l'autre avec une méthode approchée.

La résolution exacte L'évaluation de la suppression des contraintes dont l'écart d'indice est supérieur à N^*_{inc} ne peut se faire que sur des instances dont le nombre de trains est supérieur à N^*_{inc}. Or la résolution exacte d'un instance réelle à 20 trains par exemple, nécessite 5 jours de traitement. Pour réaliser l'expérimention avec la résolution exacte, nous avons donc considéré une instance plus simple, obtenue en diminuant le nombre de conflits (100 conflits contre 200 au minimum pour les instances réelles), il s'agissait à l'origine de l'instance réelle TGV-MA. Cette réduction des conflits a permis d'avoir une valeur de bt_{inc} plus faible. Le tableau 5.5 présente le gain obtenu grâce à cette information. L'optimisation a été faite pour 14 trains, le nombre maximum de trains pouvant être en conflit, sur cette instance particulière, est de 8 ($N^*_{inc} = 8, bt_{inc} = 283s$).

La résolution approchée Avec ce type de résolution, nous avons pu tester tous les types d'instances présentées en section 3.5 avec 100 trains. Les résultats sont présentés sur le tableau 5.5. L'algorithme employé est la métaheuristique LNS+VNS présentée en section 4.4.3, avec les caractéristiques suivantes :

- l'algorithme s'arrête une fois que toutes les variables ont été relâchées,
- la taille du voisinage est fixée à 1,
- le nombre de trains est fixé à 100,
- la valeur indiquée correspond à la moyenne sur l'ensemble des résolutions.

L'évalution est réalisée à partir des deux modèles suivants :

- M_{2345} : le modèle complet sans la contrainte NGBS($M^1_{ref} = M_{2345}$).
- M_{245} : le modèle complet sans la contrainte NGBS et ni la diminution du nombre de contraintes (M_{245}).

L'élimination des contraintes d'incompatibilité et des variables bornes d'incompatibilité inutiles permet un gain significatif lors de la résolution du problème. Il est de l'ordre de 17% sur le temps de résolution exacte et de 18% pour la résolution approchée. Ces éliminations permettent également de diminuer l'espace mémoire : gain de 20% pour l'instance à 14 trains, et de 75% pour

résolution instance	exacte instance à 14 trains		approchée moyenne instances à 100 trains	
modèle	M_{245}	M_{2345}	M_{245}	M_{2345}
temps de résolution :	262 s	192 s	380.33 s	351.5 s
nombre de contraintes	91	56	4 950	980

TAB. 5.4 – Impact de l'élimination des contraintes et variables inutiles en moyenne et sur une instance plus simple

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
temps de résolution	M_{245}	180 s	249 s	356 s	459 s	430 s	608 s
	M_{2345}	168 s	233 s	310 s	423 s	398 s	567 s
valeur de N^*_{inc}		15	18	21	17	20	21
réduction du nombre de contraintes		85%	82%	79%	82%	80%	79%

TAB. 5.5 – Impact de l'élimination des contraintes et variables inutiles sur des instances à 100 trains

les instances à 100 trains. En ce qui concerne le nombre de contraintes posées, pour l'instance utilisée dans la résolution exacte il est de 38% plus faible et pour l'instance de la résolution approchée il est de 80% plus faible.

Cette amélioration est tout à fait utile à la résolution, le modèle de référence n'est donc pas modifié, et reste à :$M^2_{ref} = M_{2345}$.

Ajout des coupes

Les sections 3.3.2 et 4.2.1 ont présenté la définition et les paramètres des coupes utilisées dans notre modélisation, nous allons ici étudier leur impact sur la résolution. Les coupes définies consistent à imposer des écarts minimum (m^*_s) sur les dates d'entrée de deux trains ayant un écart d'indices s. Les écarts temporels m^*_s sont obtenus par la résolution de problèmes (modèle M_{235}) à s trains pour de petites valeurs de s (relativement peu de trains).

Le nombre maximum de trains considérés a été fixé à 7, ce nombre représentant le meilleur ratio entre le temps de calcul nécessaire à la résolution des problèmes pour obtenir les coupes et le gain obtenu sur la résolution complète du problème (cf. tableau 5.6).

Pour évaluer l'apport des coupes, nous avons expérimenté l'utilisation de deux modèles :

- le modèle M_{235} qui ne contient pas les coupes,
- le modèle M_{2345} qui est le même modèle mais avec les coupes.

Les résultats des expérimentations sont repris sur le tableau 5.6.

Comme ce tableau (cf. tableau 5.6) le montre, les coupes permettent une diminution conséquente du temps de résolution, de l'ordre de 81%.

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
12 trains	M_{235}	5.32 s	16.05 s	69.63 s	91.55 s	125.43 s	10 876 s
	M_{2345}	2.15 s	7.26 s	16.78 s	55.47 s	71.6 s	2305.34 s
moyenne M_{235}/M_{2345}		84%	88%	82%	76%	81%	71%

TAB. 5.6 – Impacts des coupes sur la résolution

L'ajout des coupes s'avère très profitable à la résolution, cette amélioration sera donc gardée pour la suite des expériementations et le modèle retenue : M_{2345}.

Ajout de la borne supérieure

À la section 3.3.2, nous avons définit une méthode d'obtention d'une borne de notre problème P. Cette méthode consistait à resoudre un problème P' plus simple dont la solution est aussi une solution valide du problème P. Les expérimentation de la section 4.2.2 ont abouti à fixer le nombre de trains du problème P' à 7.

Les deux modèles utilisés pour évaluer l'apport d'une borne supérieure sont M_{234} (modèle sans cette borne) et M_{2345}. Les résultats des expérimentations sont synthétisés par le tableau 5.7 Les résolutions sont effectuées sur l'ensemble des instances et sur des tailles de 7 à 12 trains.

type instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
12 trains	M_{234}	3.67 s	9.31 s	18.52 s	102.32 s	134.43 s	2424.59 s
	M_{2345}	2.15 s	7.26 s	16.78 s	55.47 s	125.43 s	2305.34 s
moyenne M_{234}/M_{2345}		28%	21%	40%	62%	18%	11%

TAB. 5.7 – Impacts de la borne sur la résolution

L'ajout de la borne supérieure permet une diminution du temps de résolution de l'ordre de 30% en moyenne. Cette borne est très performante sur l'instance TGV-MA, car elle est très proche de la solution optimale. La borne supérieure obtenue pour l'instance TGV-MA est la même que pour l'instance TGV-CL-MA. Afin de pouvoir évaluer la qualité de la borne supérieure par rapport aux solutions optimales obtenues, la figure 5.1 représente l'écart entre la borne supérieure et la solution optimale où bs représente la valeur obtenue par la borne supérieure et $m*$ la valeur optimale. La courbe représente la valeur moyenne de cet écart pour l'ensemble des instances réelles et aléatoires. La borne supérieure donne une solution en moyenne 25% plus importante que la solution optimale, qui plus est cet écart (en proportion) diminue fortement avec l'augmentation du nombre de trains, comme l'a montré le tableau 4.3 de la section 4.2.2. Cette tendance vient conforter la remarque que nous faisions concernant l'extension à 100 trains de cette borne : les résultats obtenus sont proches (en proportion) de ceux trouvés par Delorme [Delorme et al.01], avec une moyenne de 96 trains pouvant passer sur l'infrastructure en moins d'une heure.

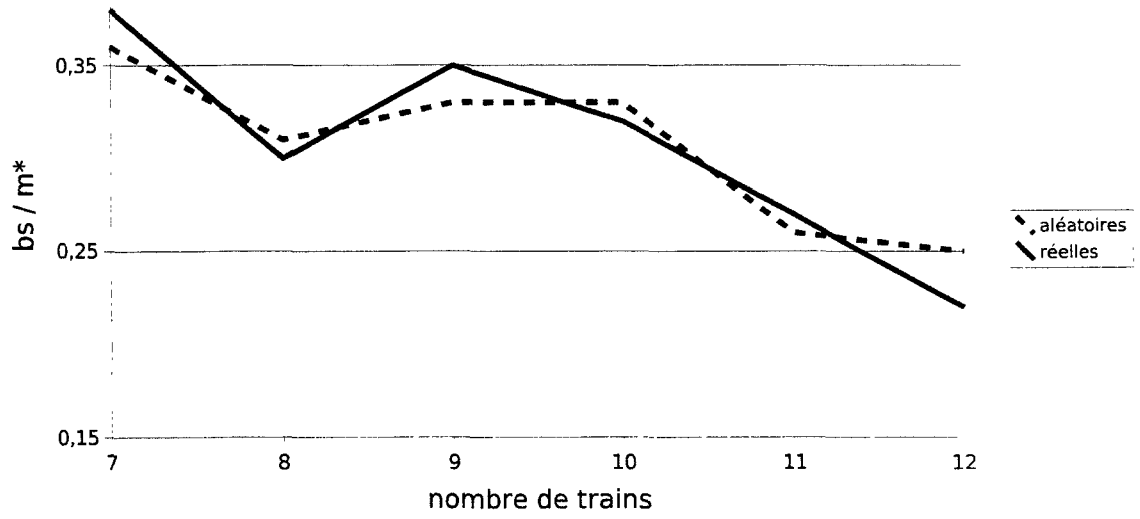

FIG. 5.1 – Écart entre la borne supérieure (bs) et la solution optimale (m*)

Les résultats de ces expérimentations ont montré que la borne supérieure pouvait contribuer favorablement à l'efficacité des méthodes de résolution. Cette amélioration a donc été retenue dans le modèle : $M^4_{ref} = M_{2345}$.

Contraintes d'incompatibilité

Dans la modélisation proposée, les contraintes d'incompatibilités (inc_1 contrainte 3.6) sont des contraintes disjonctives. Une seconde implémentation des contraintes a été définie, nommée inc_2 (contrainte 3.22), qui utilise une variable d'écart $\delta_{i,j}$ prennant ses valeurs en dehors de l'intervalle définit par les variables bornes $\underline{t_{ij}}, \overline{t_{ij}}$.

Cette seconde implémentation permet de propager les affectations de valeurs des variables bornes d'incompatibilités ($\underline{t_{ij}}, \overline{t_{ij}}$) et dates de départs ($st_i$), sans qu'il soit nécessaire d'explorer les alternatives des contraintes disjonctives.

Nous allons présenter les résultats des expérimentations dont l'objectif etaient d'évaluer l'apport de l'implémenation inc_2 par rapport à l'implémentation inc_1 des contraintes d'incompatibilité.

Les modèles utilisés pour cette évaluation sont :

- M_{345} où les incompatibilités sont implémentées avec la contrainte inc_1,
- M_{2345} où les incompatibilités sont implémentées avec la contrainte inc_2,

Les résultats présentés sur le tableau 5.8 sont des ratio obtenus sur chaque type d'instance entre ces deux modèles.

Les tailles d'instances traitées sont de $N = 7, \ldots 12$ trains, pour les mêmes raisons que précé-

demment, à savoir un temps de résolution trop important pour des tailles supérieures. Deux types de ratio sont présentés :

- un ratio sur le temps de résolution (temps),
- un ratio sur le nombre de nœuds explorés pour trouver la solution optimale (# nœuds).

type instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
12 trains	M_{345}	5.83 s	16.89 s	35.8 s	152.14 s	280.78 s	7 068 s
	M_{2345}	2.15 s	7.26 s	16.78 s	55.47 s	125.43 s	2305.34 s
moyenne M_{345}/M_{2345}		58%	67%	60%	51%	53%	43%

TAB. 5.8 – Comparaison des deux implémentations des contraintes d'incompatibilités

L'intérêt de la nouvelle implémentation de la contrainte d'incompatibilité inc_2 est ici démontré. En moyenne, l'augmentation du nombre de nœuds explorés entre $in2$ et $inc1$ est de 62% et l'augmentation du temps de résolution est de 52%. La différence de gain obtenue entre les instances aléatoires et réelles peut s'expliquer par un nombre moins important d'intervalles d'incompatibilités identiques pour les instances aléatoires, ainsi moins de valeurs de variables ont le même impact, ce qui rend le choix de valeurs plus aisé.

5.1.3 Conclusion

Les raffinements apportés au niveau de la modélisation ont permis une diminution conséquente du temps de calcul. Le cumul de toutes les améliorations aboutit à diviser le temps de résolution par 15 en moyenne. Pour les instances réelles la réduction est un peu plus faible : 10 fois. Cette remarque peut s'expliquer par le fait que les instances réelles sont plus difficiles à résoudre à cause d'un grand nombre de parcours impliquant des intervalles d'incompatibilités similaires. L'amélioration apportant le plus est l'incorporation des coupes avec une diminution de 93% des temps de calcul, la contrainte $NGBS$ (éliminer des symétries) quant à elle n'a pas permis de gain sur la résolution.

À la lecture des résultats présentés ici, le meilleur modèle est le modèle M_{2345} incorporant les améliorations :

- diminution du nombre de contraintes,
- coupes,
- borne supérieure,
- réécriture de la contrainte d'incompatibilités.

Ce sera donc ce modèle que nous utiliserons pour la résolution approchée du problème ferroviaire traité. La résolution exacte a montré ses limites, par exemple, le temps de résolution de l'instance TGV-CL-MA à 12 trains nécessite plus de 2 heures, or une instance à 12 trains n'occupe l'infrastructure que sur une période de 5 minutes. Ceci est très loin de notre objectif d'occupation d'une

heure. Aussi la section suivante présente les résultats obtenus par les heuristiques et métaheuristiques utilisées pour la résolution de notre problème sur des tailles d'instances correspondant à notre objectif.

5.2 Résolution approchée

Les résultats exposés dans cette section concernent les algorithmes de résolution approchée avec la modélisation M_{2345}. Ils reprennent et complètent ceux de Degoutin [Degoutin et al.05b], [Degoutin05] et [Degoutin et al.05a].

5.2.1 DDS

Le but de cette section est d'évaluer la méthode DDS (Depth-bounded Discrepancy Search) en tant qu'heuristique (nombre de divergence limité), ainsi que le calibrage de ses paramètres. Cet algorithme de recherche à arborescence tronquée a été expliqué en section 2.6.2, il s'agit d'un algorithme permettant d'explorer en priorité les nœuds de l'arbre de recherche présentant le moins de divergences avec l'heuristique de choix de valeurs. Les valeurs des paramètres pour cet algorithme sont celles proposées en section 4.4.1, c'est à dire :

- $depth$: 4, 6 ou 8,
- $width$: 6, 8 et 10,
- $disc$: 8, 10 et 12.

Ces valeurs de paramètres impliquent 25 combinaisons possibles, combinaisons testées sur l'ensemble des instances. Nos premiers résultats ont montré que les instances aléatoires sont plus rapides à résoudre que les instances réelles, aussi une distinction entre les deux est faite pour la présentation des résultats. En effet, le nombre d'échecs de l'heuristique d'ordre devrait être moins important sur les instances aléatoires comme nous avons pu le voir sur les figures 4.2 et 4.3 de la section 4.1.2 que sur les instances réelles. Ainsi les meilleurs valeurs de paramètre ne devraient pas être les mêmes, et les tailles d'instances aléatoires que nous pourrons résoudre pourront être plus importantes.

Instances réelles

Sur les instances réelles, le nombre de trains est compris entre 8 et 12 compte tenu des temps de résolution fixés (moins d'une heure). Le diagramme de la figure 5.2 représente le gain obtenu sur le temps de résolution par l'heuristique DDS en comparaison avec la résolution exacte. La figure 5.3, quant à elle, compare la qualité de la solution obtenue par l'algorithme DDS par rapport à la solution optimale. Afin de ne pas surcharger les figures, seules les 9 combinaisons de paramètres les plus intéressantes sont présentées.

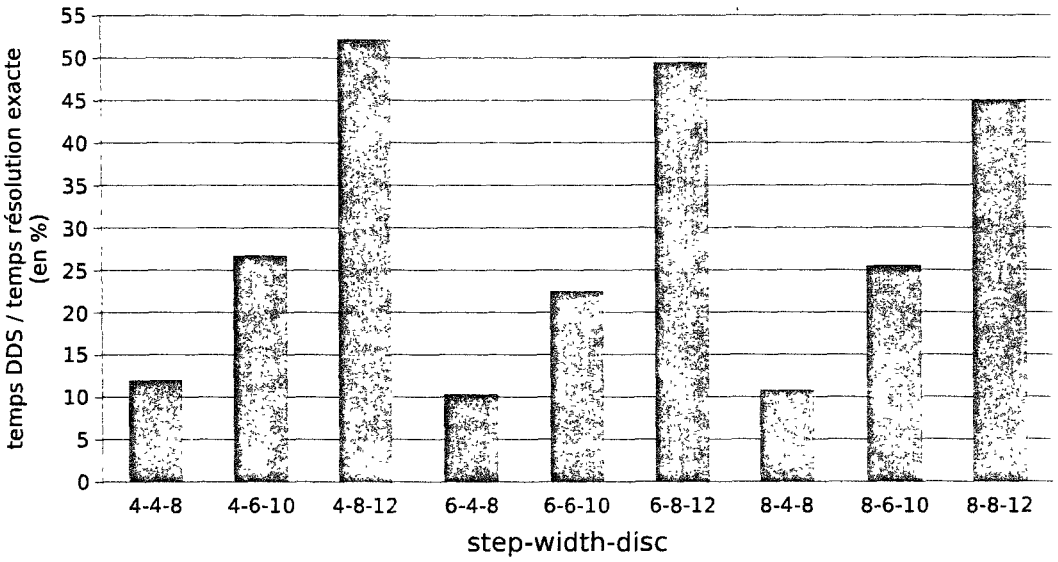

FIG. 5.2 – Impact des paramètres DDS sur le temps de résolution (instances réelles)

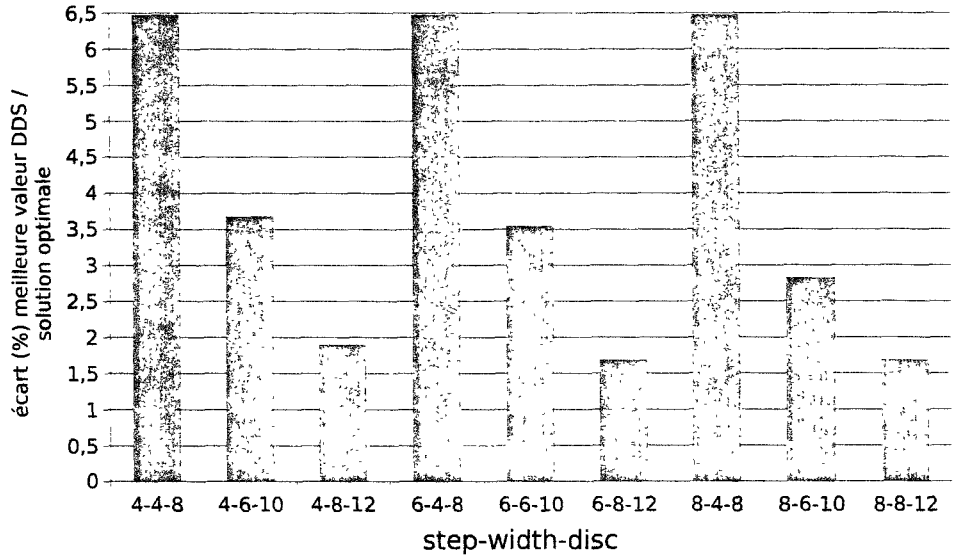

FIG. 5.3 – Impact des paramètres DDS sur la qualité des solutions (instances réelles)

Ces résultats montrent, comme on pouvait s'y attendre, qu'en général plus le temps de résolution est grand, meilleure est la solution. Aussi un compromis entre la qualité de la solution et le temps de résolution doit être trouvé, pour cela nous considérerons qu'une solution à moins de 2% de la solution optimale est une solution de bonne qualité. Selon ce compromis, la meilleure combinaison des paramètres pour DDS, sur les instances réelles, est la suivante :

- $step = 6$,
- $width = 8$,
- $disc = 12$.

Malgré l'augmentation du nombre de trains, la qualité de la solution et le gain en temps de résolution restent identiques, comme le montre le tableau 5.9 où les valeurs présentées correspondent aux combinaisons de paramètres $6 - 8 - 12$.

type instance		TGV-MA	CL-MA	TGV-CL-MA
temps DDS/exact	8 trains	-51%	-54%	-58%
	12 trains	-49%	-56%	-59%
qualité solution DDS/exact	8 trains	1.8%	2.1%	1.8%
	12 trains	1.9%	2.1%	1.8%

TAB. 5.9 – Temps de résolution et qualité de résolution pour des tailles d'instances différentes

Instances aléatoires

Étant donné que les instances aléatoires sont plus rapides à résoudre que les instances réelles, le nombre de trains a été fixé entre 8 et 14. Comme pour les instances réelles, la figure 5.4 représente le gain obtenu sur le temps de résolution par l'heuristique DDS en comparaison avec la résolution exacte. Le diagramme de la figure 5.5, quant à lui, compare la qualité de la solution obtenue par l'algorithme DDS par rapport à la solution optimale. Ici aussi, seules les 9 combinaisons de paramètres les plus intéressantes sont présentées. Les « très bons résultats » obtenus sur ces instances par l'heuristique de choix de valeurs $eval_r$ ont conduit à diminuer le nombre de divergences à 4 (cela implique un parcours moins important de l'arbre de recherche).

Par rapport aux instances réelles, ces résultats présentent un moins bon ratio pour le temps de résolution pouvant s'expliquer par le fait que les instances aléatoires se résolvent plus rapidement que les instances réelles. Ceci implique que le temps de traitement nécessaire à la mise en œuvre de DDS est en proportion plus important que sur les instances réelles, comme tend à le montrer le tableau 5.10. Par contre le ratio pour la comparaison de la qualité des solutions est bien meilleur, grâce à l'heuristique d'ordre $eval_r$ qui s'est révélée plus efficace sur ces instances, ainsi le nombre de divergences est plus faible. Pour les instances aléatoires et en suivant le même critère que pour les instances réelles, les meilleurs paramètres sont :

- $step = 6$,
- $width = 6$,
- $disc = 6$.

Comme pour les instances réelles, les ratio de la qualité de la solution sont identiques malgré l'évolution du nombre de trains, comme le montre le tableau 5.10 où les valeurs présentées correspondent aux combinaisons de paramètres $6 - 6 - 6$.

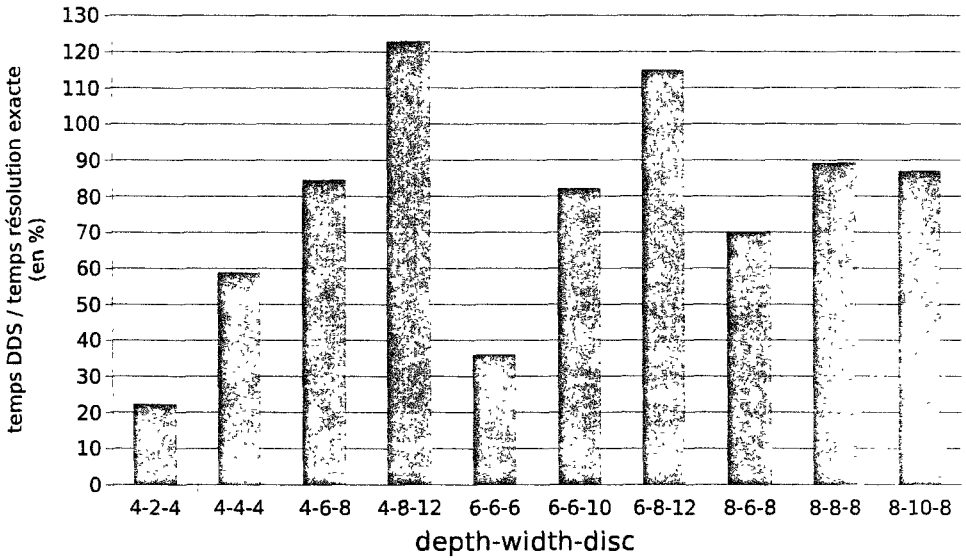

FIG. 5.4 – Impact des paramètres DDS sur le temps de résolution (instances aléatoires)

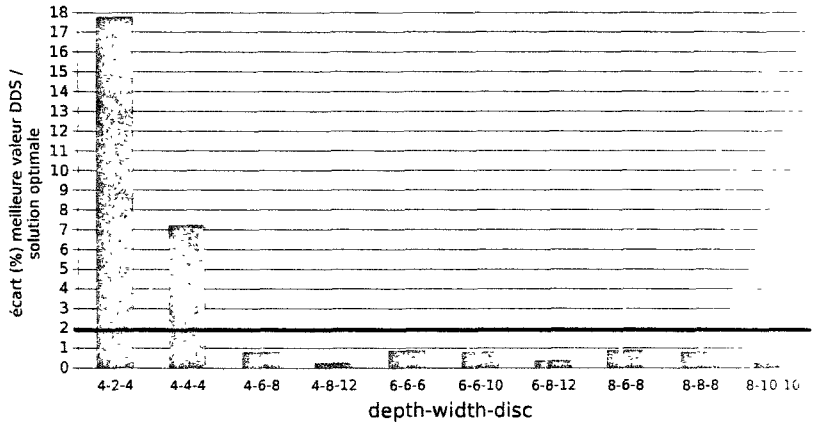

FIG. 5.5 – Impact des paramètres DDS sur la qualité des solutions (instances aléatoires)

type instance		aléa1	aléa2	aléa3
temps DDS/exact	8 trains	-35%	-32%	-31%
	12 trains	-40%	-37%	-36%
qualité solution DDS/exact	8 trains	0.9%	0.7%	0.8%
	12 trains	0.9%	0.7%	0.8%

TAB. 5.10 – Temps de résolution et qualité de résolution pour des tailles d'instances différentes

5.2.2 Recherche à voisinage large

Cette section présente les différents résultats obtenus grâce à la métaheuristique LNS (Large Neighborhood Search) expliquée en section 2.7.2 et dont le fonctionnement pour le problème fer-

roviaire est indiqué en section 4.4.2. Le nombre de trains des instances à résoudre est fixé à 100, valeur nous permettant de nous situer au niveau des meilleures valeurs obtenues par Xavier Delorme [Delorme03] : les meilleures solutions obtenues en 2001 par la métaheuristique GRASP [Delorme et al.01] étant de 97 trains parcourant l'infrastructure en une heure. À noter qu'un gain, sur notre critère, même de 1% permettrait de pouvoir insérer 1 train supplémentaire sur l'infrastructure, ce qui est très intéressant.

Choix de la solution initiale de bonne qualité

En section 4.4.2, trois méthodes d'obtention d'une solution initiale ont été sélectionnées :
- la borne supérieure obtenue par la répétition d'une même séquence de parcours sera nommée "séquence",
- la solution trouvée par l'algorithme proposé par Delorme [Delorme03] sera nommée "GRASP",
- une solution de mauvaise qualité composée uniquement de trains empruntant le même parcours sera nommée "MQ".

Les expérimentations présentées ici ont tout d'abord pour objectif de comparer les deux première méthodes. A priori, la métaheuristique GRASP devrait avoir une meilleure efficacité. En effet la solution donnée par la borne supérieure est une répétition de séquences de parcours. Cette comparaison doit nous permettre d'évaluer la capacité de ces motifs à bloquer la méthode LNS dans un optimum local. L'intérêt d'utiliser une solution initiale de GRASP est donc double :
- la solution est de bonne qualité,
- la solution générée ne doit présenter que peu de motifs dû au caractère aléatoire de GRASP.

Avec l'utilisation de la méthode "MQ", le second objectif de ces expérimentations est d'évaluer l'impact des solutions initiales sur la résolution.

Le temps alloué à la résolution approchée a été fixé à 30 minutes au maximum. Le temps nécessaires à l'obtention des solutions initiales n'est pas le même suivant la méthode retenue :
- 15 minutes sont nécessaires pour obtenir une solution satisfaisante avec GRASP,
- 2 minutes sont nécessaires pour obtenir la solution proposée par la séquence.

Ainsi le temps de résolution accordé à l'algorithme LNS couplé à l'heuristique DDS a été fixé à 15 minutes pour les deux types de solutions initiales, afin de pouvoir comparer sous les mêmes conditions ces 2 méthodes.

Les valeurs des solutions initiales obtenues par ces deux méthodes sont décrites sur le tableau 5.11. Ils mettent en évidence la diversité des qualités de solutions initiales obtenues par GRASP. La méthode GRASP utilise un modèle de Set Packing, les solutions sont donc des valeurs de variables booléennes. Pour être utilisées comme solution de notre modèle CSP, les solutions de GRASP doivent être transformées en valeurs pour les variables parcours et dates d'entrée. C'est pour cette raison que dans les expérimentations, nous n'avons considéré que les instances réelles pour lesquelles il est possible de définir ce mécanisme de transformation. D'autre part, cet algorithme comporte une composante aléatoire. Les résultats que nous présentons pour chaque instance

correspondent donc à une moyenne sur 10 lancements de l'algorithme. Le tableau 5.12 compare les résultats obtenus en utilisant les deux solutions initiales proposées (GRASP et séquence).

type d'instances	Sol. init.	makespan		
		moyenne	min	max
TGV-MA	GRASP	3 960	3 720	4 342
	séquence	3 872	3 872	3 872
CL-MA	GRASP	4 120	3 940	4 340
	séquence	3 808	3 808	3 808
TGV-CL-MA	GRASP	3 880	3 560	4 189
	séquence	3 872	3 872	3 872

TAB. 5.11 – Valeurs des solutions initiales

type d'instances	Sol. init.	makespan		
		moyenne	min	max
TGV-MA	GRASP	3 407	3 373	3 455
	séquence	3 369	3 369	3 369
CL-MA	GRASP	3 531	3 464	3 605
	séquence	3 532	3 532	3 532
TGV-CL-MA	GRASP	3 396	3 338	3 473
	séquence	3 367	3 367	3 367

TAB. 5.12 – Comparaison des solutions finales selon les solutions initiales

Globalement, on constate que LNS avec la solution initiale répétant une même séquence de trains présente les meilleures performances. On remarque que ponctuellement, des solutions de meilleure qualité sont obtenues avec la solution initiale de la méthode GRASP.

Pour les solutions initiales provenant de GRASP, la meilleure solution finale atteinte correspond à la solution initiale de GRASP, soit : 101 trains. Nous nous sommes intéressés à la diversité des parcours retenus dans les solutions de GRASP. Ces solutions possèdent en moyenne 5 parcours différents, le maximum étant de 7. Il s'avère donc que ce nombre est comparable à celui de la solution obtenue par la répétition de même séquence. Ceci tend également à montrer que seules des combinaisons de petits ensembles de parcours aboutissent aux séquences de trains les plus compactes.

Enfin, un autre aspect négatif de la méthode GRASP est le temps de calcul nécessaire à l'obtention de la solution. Ce temps est de 15 minutes, ce qui est bien supérieur aux 2 minutes nécessaires à la méthode par séquence de trains. Il s'avère donc plus intéressant de garder la solution initiale obtenue par la borne supérieure.

Résolution des sous-problèmes

Un autre paramètre du schéma LNS, est la méthode utilisée dans la phase de ré-optimisation des voisinages. L'objectif de cette section est de comparer deux méthodes pour la phase de réoptimisation de LNS :

- l'algorithme de résolution exacte,
- l'heuristique DDS.

L'utilisation d'une méthode de résolution approchée, comme DDS, pour la résolution des sous-problèmes générés par LNS peut apporter deux intérêts (fortement liés) :

1. augmenter la taille du voisinage,
2. diminuer le temps de résolution tout en obtenant des solutions de même qualité.

Comme vu précédemment en section 5.2.1, l'heuristique DDS implémentée est efficace pour la résolution de problèmes de petites tailles. Il est donc envisageable d'intégrer cette heuristique à la métaheuristique LNS pour la résolution des sous-problèmes (réoptimisation des voisinages). De plus l'heuristique de choix de parcours implémentée est susceptible d'être plus efficace car les trains, autour du voisinage considéré, sont déjà affectés, le nombre de divergences devrait donc être moindre que pour la résolution des instances de petite taille. Le choix de valeurs des paramètres associés à DDS a été borné par les expérimentations effectuées en section 5.2.1. Le tableau 5.13 reprend les principales combinaisons de paramètres de DDS testées. Les résultats qui y sont présentés sont les valeurs de makespan obtenus pour la combinaison de paramètres indiqués, sur l'ensemble des instances et pour un temps de résolution des sous-problèmes fixé à 15 minutes.

type d'instance / paramètres $step - width - disc$	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
4-6-8	3 610s	4 010s	3 021s	3 258s	3 602s	3 258s
6-6-6	3 595s	4 010s	3 021s	3 258s	3 602s	3 258s
6-8-10	3 601s	4 010s	3 021s	3 258s	3 602s	3 258s

TAB. 5.13 – Makespan selon les paramètres de DDS

Nous pouvons remarquer que les valeurs des paramètres n'ont pas un grand impact sur la résolution. Néanmoins, la meilleure combinaison de valeurs est :

- $depth$: 6
- $width$: 6
- $disc$: 6

En fait, de grandes (supérieure à 12 pour la divergence) ou faibles (inférieure à 4 pour la divergence) valeurs pour ces paramètres vont influer sur la qualité de la solution finale, mais le changement d'une ou deux unités par rapport à notre jeu de valeurs n'a quasiment pas d'effet sur la résolution.

Une fois le calibrage des paramètres de DDS réalisé, l'expérimentation suivante consiste à comparer l'utilisation de la méthode approchée DDS avec la méthode exacte de B&B (cf. section 4.3) pour la résolution des sous-problèmes du schéma LNS. Les résultats de cette expérimentation sont repris dans le tableau 5.13. Les valeurs indiquées correspondent au ratio exposé en introduction de ce chapitre. Le temps de résolution a été fixé à 15 minutes.

type d'instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA	moyenne
exacte / DDS	0.75%	0.8%	0.9%	1.0%	0.9%	1%	0.9%

TAB. 5.14 – Comparaison des valeurs de solutions obtenues en utilisant DDS ou une résolution exacte

Comme le tableau 5.14 l'indique, l'écart entre la résolution exacte et l'heuristique DDS est faible, mais à l'avantage de l'heuristique DDS. Cette meilleure qualité des solutions de l'algorithme DDS peut s'expliquer par des résolutions plus rapides des sous-problèmes (cf. section 5.2.1), ce qui permet une exploration plus importante de l'espace de recherche. Ainsi, pour la suite de nos expérimentations, l'heuristique DDS sera utilisée.

5.2.3 Recherche à voisinage variable

Dans cette section les résultats obtenus par la combinaison de la métaheuristique de recherche à voisinage large (LNS) et de la métaheuristique de recherche à voisinage variable nommée VNS sont détaillés et analysés. Le fonctionnement de la métaheuristique VNS a été expliqué en section 2.7.3 et 4.4.3. La combinaison de ces 2 métaheuristiques est assez simple, à chaque fois qu'un voisinage a été testé sur l'ensemble des variables, deux cas peuvent se présenter :

- si la solution a été améliorée, alors la taille de voisinage ne change pas,
- si elle n'a pas été améliorée, il convient alors d'augmenter la taille du voisinage.

Dans la suite, les résultats sur les différents type et évolution de taille de voisinage et sur la comparaison de l'algorithme LNS seul et de la combinaison LNS+VNS sont proposés.

Type et évolution de la taille du voisinage

Cette section a pour but d'étudier l'impact du type de voisinage sur la résolution. Nous aurions pu nous attendre à une différence entre l'algorithme LNS seul et l'algorithme LNS+VNS, du fait que justement VNS se focalise sur les types de voisinages, mais ce n'est pas le cas. Ainsi l'impact des types et évolution de voisinage étant identiques pour l'algorithme LNS et pour l'algorithme LNS+VNS, les résultats présentés ici correspondent à la moyenne sur ces deux algorithmes.

À chaque phase LNS les variables relâchées correspondent à des séquences de trains successifs. Le passage d'un voisinage à l'autre s'effectue soit de manière aléatoire (*aléa*), soit selon un ordre

chronologique (*chrono*). Une fois toutes les variables relâchées, si le voisinage doit être modifié sa taille est alors augmentée, soit de 1 unité (Δ_1), soit de 2 unités (Δ_2), soit de 3 unités (Δ_3).

Le tableau 5.15 présente les impacts de l'augmentation du pas de la taille du voisinage (cf. section 4.4.2) pour les deux types de voisinage (*chrono* et *aléa*). Les valeurs indiquées correspondent à la moyenne sur les algorithmes de résolution LNS et LNS+VNS pour l'ensemble des instances. Le temps de résolution est fixé à 15 minutes. Pour le voisinage aléatoire, les résultats présentés correspondent à la moyenne sur 10 exécutions.

type d'instance	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
Δ_1 / Δ_2	1.5%	2%	1.5%	0.5%	0.5%	-0.5%
Δ_1 / Δ_3	-1%	0.5%	-1.5%	-1.5%	-2.5%	-3.5%

TAB. 5.15 – Ratio entre les solutions obtenues selon le pas d'augmentation du voisinage.

Le tableau 5.15 présente les ratios entre les solutions suivant le pas d'augmentation de la taille du voisinage. L'augmentation de la taille de voisinage de deux unités (Δ_2) permet de trouver de meilleures solutions en moyenne et principalement sur les instances aléatoires. Sur les instances aléatoires, la taille du voisinage est au final bien plus importante que celle des instances réelles ($\max Neigh_{size} = 14$ pour les instances aléatoires et $\max Neigh_{size} = 8$ pour les réelles), ce qui permet une exploration bien plus conséquente de l'espace de recherche. Par contre, augmenter la taille des voisinages par pas de 3 rend bien moins efficace la résolution. En effet, la taille des voisinages augmente très fortement, ce qui implique un temps de résolution par réoptimisation très important (cf. les tailles d'instance traitées en résolution exacte en section 5.1.2), et donc trop peu de voisinages peuvent être explorés.

Le tableau 5.16 présente les impacts des différents types de voisinage (cf. section 4.4.2). Les valeurs indiquées correspondent à la moyenne sur les algorithmes de résolution LNS et LNS+VNS pour l'ensemble des instances. Le temps de résolution est fixé à 15 minutes. Pour le voisinage aléatoire, les résultats présentés correspondent à la moyenne sur 10 lancements.

type d'instance	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
chrono / *aléa*	-3%	-5%	- 4%	-9%	-12%	-15%

TAB. 5.16 – Ratio entre les solutions obtenues avec les différents types de voisinages

Le tableau 5.16 présente le ratio des valeurs de makespan selon l'ordre choisi pour les changements de voisinage. Le choix aléatoire du changement de voisinage n'est a priori pas intéressant, les valeurs étant bien supérieures (donc moins bonnes) à celles obtenues via un changement de voisinage fixé. Le tableau 5.17 vient compléter cette analyse. Ce tableau présente la meilleure solution

obtenue sur les 10 lancements aléatoires pour une instance donnée, comparée à la solution obtenue avec un voisinage évoluant dans l'ordre chronologique. Ainsi, même si en moyenne les valeurs de makespan sont moins bonnes, les meilleures solutions sont atteintes avec un changement aléatoire.

type d'instance	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
chrono	3 048s	3 769s	2 775s	3 221s	3 550s	3214s
aléa	3 048s	3 769s	2 756s	3 212s	3 550s	3212s

TAB. 5.17 – Meilleures valeurs selon le type de voisinage

Aux vues des résultats sur les types de voisinages, il s'avère qu'augmenter le voisinage par pas de 2 (Δ_2) trains est une bonne stratégie, principalement pour les instances aléatoires. Il est plus intéressant d'utiliser un voisinage changeant de manière chronologique (*chrono*), qu'aléatoire.

Apport de la recherche à voisinage variable

Cette section s'intéresse à l'impact de l'algorithme de recherche à voisinage variable (VNS) pour guider la recherche de la métaheuristique LNS. Deux cas sont présentés :

- la résolution démarre grâce à une bonne solution initiale (séquence), cette solution étant la répétition d'une même séquence de parcours (équivalente à la borne supérieure),
- la résolution débute avec une solution de mauvaise qualité (MQ), les trains utilisent tous le même parcours, ce parcours étant le moins bon préconisé par l'heuristique de choix de valeurs statique ($eval_{statique}$ équation 4.1). Ce type de solution initiale permet d'évaluer l'impact de cette solution sur la qualité de la solution finale.

Les résultats des deux tableaux 5.18 et 5.20 permettent de comparer les solutions obtenues par les algorithmes LNS seul et LNS+VNS, sur l'ensemble des types de voisinage, pour des temps de résolution (15 minutes) et des combinaisons de paramètres identiques. Suivant les résultats obtenus en section 5.2.3, deux résolutions ont été effectuées, l'une avec un pas d'augmentation du voisinage a fixé à 1 (Δ_1) et l'autre à 2 (Δ_2) . Le nombre de lancement sur les changements de voisinages aléatoire est de 5.

type d'instance	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
(LNS+VNS / LNS) séquence	-0.5%	0.5%	0%	0%	-0.5%	0%
(LNS+VNS / LNS) MQ	3%	5%	4%	2%	1.5%	2.5%

TAB. 5.18 – Ratio entre les solutions obtenues par LNS et LNS+VNS

Comme nous pouvons le remarquer sur le tableau 5.18, l'algorithme LNS seul semble obtenir des solutions de meilleure qualité, de l'ordre de 1.5% en moyenne. Mais en observant plus en

détail, cet algorithme n'est meilleur que si la solution initiale est de mauvaise qualité. Une raison pouvant l'expliquer est qu'une solution de mauvaise qualité demandera plus d'effort (plus de modifications de valeurs de variables) pour arriver à une bonne solution. LNS+VNS tend à utiliser des voisinages de plus petites tailles que LNS puisque VNS impose des conditions plus fortes (plus d'amélioration du critère) pour augmenter la taille du voisinage. Ainsi, pour une solution initiale de mauvaise qualité, l'utilisation de voisinages de grandes tailles permettra une descente plus rapide (amélioration des solutions). Pour mieux comprendre cette idée, considérons l'exemple de la figure 5.6. Les courbes indiquent les valeurs de makespan en fonction du temps et de la taille de voisinage utilisée pour une solution initiale de mauvaise qualité. La courbe pleine correspond à une taille de voisinage de 1, celle en pointillé correspond à une taille de voisinage de 4. Comme nous pouvons le voir, sur une même période de temps un voisinage de taille plus grande permettra une descente plus rapide.

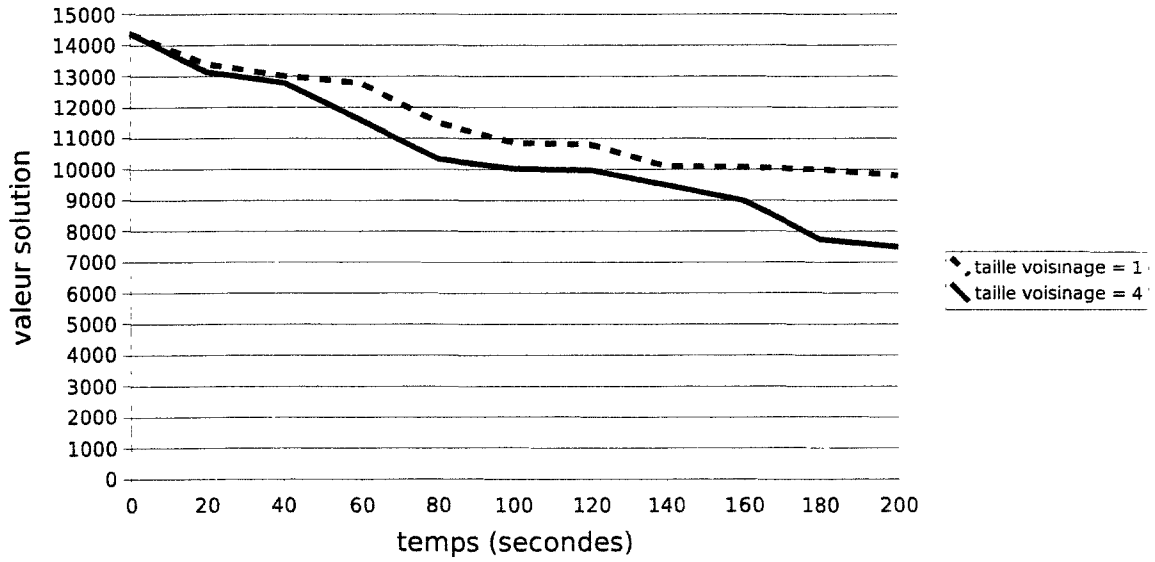

FIG. 5.6 – Évolution du critère en fonction de la taille du voisinage (solution initiale de mauvaise qualité)

C'est ce principe que tend à confirmer les résultats présentés dans le tableau 5.19.

Si la solution initiale est de bonne qualité, ces deux algorithmes sont équivalents. Aussi le tableau 5.20 permet de les départager, ce tableau présente en effet l'écart type des solutions obtenues. Ce tableau montre ainsi que l'algorithme LNS+VNS est plus stable que LNS seul et principalement pour les instances réelles. Ainsi l'implémentation de la combinaison LNS+VNS semble être moins dépendante des différents paramètres utilisés.

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
séquence	LNS+VNS	12	11	10	9	8	7
	LNS	12	11	11	9	9	7
MQ	LNS+VNS	7	9	6	5	3	2.5
	LNS	10	10	8	7	6	3

TAB. 5.19 – Taille moyenne des voisinages à la fin de la résolution (en nombre de parcours relachées)

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
séquence	LNS+VNS	300	450	380	10	11	18
	LNS	350	451	390	15	12	21
MQ	LNS+VNS	1206	1045	1439	150	167	178
	LNS	1370	1134	1602	249	478	622

TAB. 5.20 – Écart type des solutions obtenues par LNS et LNS+VNS

Impact de la qualité des solutions initiales

Suite à l'analyse des résultats obtenus par rapport aux solutions initiales de bonne qualité de la section 5.2.2, il a été mis en évidence qu'il vallait mieux prendre celle générée par la répétition d'une même séquence de parcours que celle provenant de GRASP. Le but de cette section est de comparer les solutions obtenues avec une solution initiale de bonne qualité et une solution initiale de mauvaise qualité. Cette comparaison permettra de déterminer l'importance de la qualité de la solution initiale sur la solution finale des algorithmes de résolution approchée. Pour ce faire, le tableau 5.21 indique les ratio entre les solutions finales selon la qualité de la solution de départ. Les deux types de solutions initiales utilisées sont les mêmes que précédemment, à savoir : séquence et MQ.

Les résultats sont proposés selon le ratio décrit en introduction de ce chapitre pour l'algorithme LNS seul et pour ce même algorithme combiné à VNS (LNS+VNS).

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
séquence/MQ	LNS+VNS	-13%	-15%	-37%	-35%	-23%	-12%
	LNS	-14%	-11%	-33%	-34%	-19%	-10%

TAB. 5.21 – Ratio entre les résolutions utilisant une bonne et mauvaise solution initiale

Meilleure est la solution initiale, meilleure est la solution finale. Cependant, la différence semble être moins importante pour l'algorithme LNS seul, comme nous l'avions remarqué précédemment l'algorithme LNS se comporte mieux que LNS+VNS lorsque la solution est de mauvaise

qualité. La différence entre les solutions finales pour les instances aléatoires est plus faible que pour les réelles, ceci pouvant s'expliquer car les instances aléatoires sont plus simples à résoudre et donc la solution initiale a un impact moins important.

Afin de mieux se rendre compte des efforts effectués par ces deux algorithmes (LNS et LNS-+VNS) vis à vis des solutions initiales, le tableau 5.22 présente les différents ratio entre les valeurs des solutions initiales et finales, chaque ratio étant la moyenne des ratio des deux métaheuristiques. L'étiquette "diff séquence" se réfère à la différence entre la solution initiale de bonne qualité et la solution finale, l'étiquette "diff MQ" se réfère à la différence entre la solution initiale de mauvaise qualité et la solution finale.

instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
diff séquence	7%	8%	10%	6%	6%	6%
diff MQ	84%	87%	87%	66%	66%	67%

TAB. 5.22 – Écart entre les solutions initiales et finales

5.2.4 Diversité des parcours

Lors de l'analyse des solutions obtenues par ces métaheuristiques, nous avons pu remarquer que les trains empruntaient très peu de parcours différents. Le tableau 5.23 en dresse le nombre suivant les instances (moyenne pour les instances aléatoires), pour des tailles d'instances à 100 trains. À noter qu'il en va de même pour les solutions obtenues sur des tailles d'instance plus faibles. Au mieux 12 parcours sur les 37 possibles sont utilisés sur les instances aléatoires, et pour les instances réelles seuls 6 parcours le sont.

Type d'instances	aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
possibles	19	26	37	19	26	37
solutions initiales	2	3	3	5	4	5
solutions finales	5	6	5	7	10	12

TAB. 5.23 – Comparaison du nombre de parcours possibles des solutions initiales et celui des solutions finales.

Ayant constaté cette faible diversité des parcours utilisés dans les solutions, nous nous sommes intéressés aux modifications opérées dans la séquence des trains. Le diagramme 5.7 présente cette répartion sur une instance TGV-MA. Dans cette figure, l'ordonnée représente le nombre de changements de valeurs pour chaque variable parcours (r_i) entre la solution initiale et la solution finale.

Cette solution a été obtenue avec l'algorithme LNS+VNS à partir d'une bonne solution initiale (séquence). Ce diagramme montre que peu, voire pas de parcours sont modifiés entre les trains d'indice 15 et 85.

La concentration des changements aux extrémités de la séquence des trains s'explique par le nombre plus réduit de contraintes pour ces trains. Il y a en quelque sorte une phase d'amorce et de désamorce des contraintes d'incompatibilité aux extrémités. L'amplitude observée de ces phases est à rapprocher de la valeur du paramètre N_{inc} ($N_{inc} = 17$ pour l'instance TGV-MA, cf. tableau 4.5 section 4.2.3). On rappelle que N_{inc} est l'approximation de l'écart d'indice où les trains n'ont plus de contraintes d'incompatibilité. Autrement dit tout train d'indice i, subira potentiellement des contraintes d'incompatibilité des $N_{inc} - 1$ trains successeurs et des $N_{inc} - 1$ trains prédécesseurs. La diversité des parcours des extrémités de la séquence peut s'interpréter comme le résultat du relâchement progressif des contraintes d'incompatibilité qui se réduisent pour les indices de trains $[1, N_{inc} - 1]$ et $[N - N_{inc} + 1, N]$.

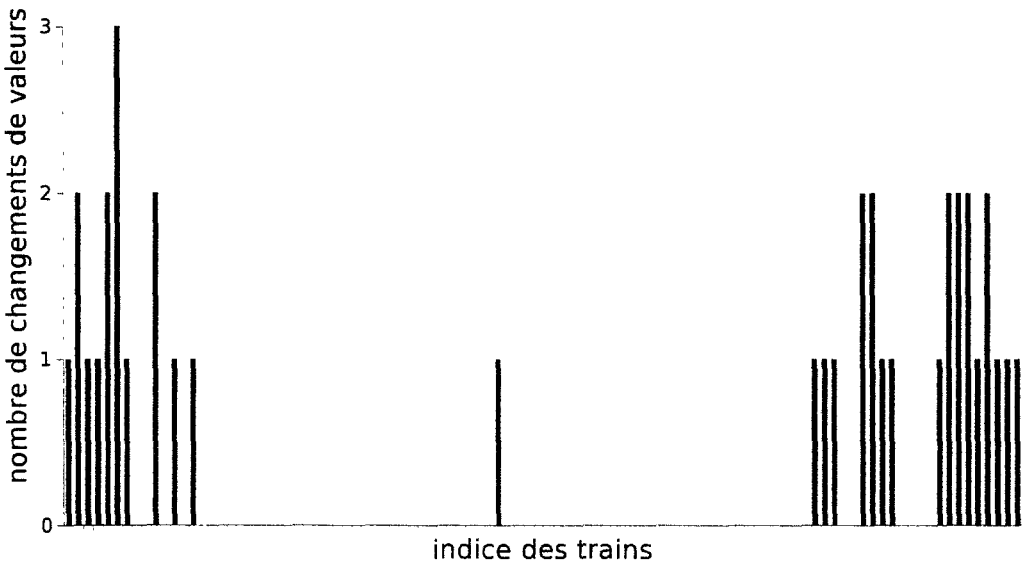

FIG. 5.7 – Nombre de modifications de valeurs des variables parcours

Aussi, afin de diversifier la recherche, nous avons implémenté une modification aux ré-optimisations de la métaheuristique. Il s'agit d'obliger l'algorithme de résolution à trouver une solution différente à celle d'origine, tout en maintenant sa qualité. Ainsi, cette diversification tendra à modifier la solution et donc à casser le motif récurrent impliqué par la solution initiale. Cette modification de l'algorithme affecte l'ensemble des variables et non pas uniquement celles non extrémales. L'implémentation a été effectuée en ajoutant pour chaque ré-optimisation un no-good interdisant à l'algorithme de trouver la solution précédente. Cela revient à rechercher des solutions meilleures

ou tout au moins équivalentes (même valeur de critère mais solution différente), solutions qui pourraient permettre d'améliorer les voisinages suivants. Le tableau 5.24 présente les résultats obtenus avec cette implémentation, avec l'étiquette LNS+VNS+SE l'algorithme utilisant cette notion de no-good. La ligne « diff nb parcours » désigne la différence entre le nombre de parcours obtenus par LNS+VNS et LNS+VNS+SE.
Comme on peut le remarquer, les solutions obtenues par la recherche de solutions équivalentes est

type d'instance		aléa1	aléa2	aléa3	TGV-MA	CL-MA	TGV-CL-MA
LNS+VNS /	diff nb parcours	0	0	1	0	1	0
LNS+VNS+SE	qualité solution	-0.5%	-1.5%	-3%	0%	-0.5%	-2%

TAB. 5.24 – Impact de la recherche de solution équivalentes

moins efficace que l'algorithme LNS+VNS initial, et ce même si la recherche de solution équivalente n'est utilisée que pour une itération de LNS (un seul passage sur l'ensemble des variables). Cette différence peut s'expliquer par le temps nécessaire à rechercher des solutions équivalentes. En effet, le temps passé pour cette recherche implique moins de temps pour tester d'autres tailles de voisinage.

5.2.5 Comparaison avec le modèle SPP/GRASP

Le tableau 5.25 présente les meilleures valeurs de *makespan* obtenues pour les instances réelles traitées, et les meilleures valeurs de chacun des ensembles de cinq instances de même type pour les instances aléatoires. Comme le présente la dernière ligne de ce tableau, ces résultats sont obtenus grâce à l'algorithme LNS+VNS sur 80% des instances, les 20% restants proviennent de l'algorithme LNS, pour tous ces meilleures résultats la solution initiale est celle répétant la même séquence de parcours.

instances	aléa1	aléa2	aléa3	TGV-MA	TGV-CL	TGV-CL-MA
st_N	3 048s	3 769s	2 756s	3 212s	3 550s	3 208s
LNS+VNS	100%	80%	60%	100%	100%	100%

TAB. 5.25 – Meilleures valeurs pour 100 trains

Ainsi, on remarque que pour l'ensemble des instances réelles, plus de 100 trains passent en moins d'une heure. Ces résultats sont les meilleurs connus pour l'instant.

Afin de pouvoir se comparer à la métaheuristique GRASP proposés par Delorme, nous avons testé nos algorithmes sur des tailles d'instances réelles allant jusqu'à 110 trains (en laissant toujours

15 minutes pour la résolution). Nous avons ainsi pu définir nos meilleures solutions par rapport au critère de Xavier Delorme, la comparaison de ces résultats, obtenus sur la même machine, est proposée dans le tableau 5.26.

type d'instance	TGV-MA	CL-MA	TGV-CL-MA
GRASP	99	86	101
LNS+VNS	108	102	110

TAB. 5.26 – Meilleures valeurs sur une heure d'occupation de l'infrastructure

5.3 Conclusion

Dans ce chapitre, les différentes améliorations du modèle et les différents algorithmes de résolution exacte et approchées proposés ont été testés et leurs performances ont été analysées. Pour réaliser ces expérimentations, nous nous sommes basés sur plusieurs instances réelles et aléatoires.

Les résultats obtenus ont permis de mettre en avant les améliorations proposées pour le modèle du problème de saturation d'infrastructure ferroviaire. L'utilisation des coupes, de la borne supérieure, la réduction du nombre de contraintes et de variables, ainsi que la ré-écriture de la contrainte d'incompatibilité ont permis des réductions importantes sur le temps de résolution, mais également sur l'espace mémoire utilisé. La fonction d'évaluation pour l'heuristique de choix de valeurs a également montré son efficacité. Il en est ressorti que le modèle le plus intéressant est le modèle M_{2345}.

Malgré ces améliorations, la résolution exacte de ce problème n'est pas possible pour les tailles d'instances à traiter. Aussi les algorithmes de résolution approchée proposés ont été utilisés et les différents paramètres possibles ont été testés. Il s'est avéré que l'algorithme de recherche à voisinage large couplé à la métaheuristique de recherche à voisinage variable et utilisant l'heuristique DDS (algorithme LNS+VNS) présente les meilleurs résultats.

Une analyse des solutions obtenues a permis de mettre en évidence que très peu de valeurs différentes de parcours sont utilisées, ainsi un algorithme de « diversification » de solution a été implémenté. Malheureusement le temps mis pour cette « diversification » est trop important par rapport au temps de résolution total et n'améliore pas les solutions.

Finalement nous avons présenté les meilleures solutions obtenues afin de comparer les performances de l'algorithme LNS+VNS avec celles connues. Pour ce faire nous avons considéré des instances de problèmes avec un plus grand nombre de trains. Les résultats montrent une amélioration de 10% en moyenne soit plus de 102 trains sur un intervalle d'une heure.

Conclusion

Les travaux présentés portent sur l'évaluation de la capacité d'infrastructures ferroviaires. Ils ont été décrits en deux axes principaux :

- la modélisation du problème au niveau d'un nœud et les améliorations apportées à ce modèle,
- les algorithmes de résolution et leur validation.

Chacun de ces axes est ici repris, leurs possibilités et limites sont précisées. Leurs évolutions et différentes perspectives sont données.

Modélisation de la capacité d'infrastructures ferroviaires

Notre travail a permis la modélisation complète du problème de l'évaluation de la capacité d'infrastructures ferroviaires au niveau d'un nœud. Le niveau de discrétisation choisi pour les passages des trains permet de considérer un niveau de détail suffisant pour l'établissement de grilles horaires sur les infrastructures considérées. La base de la modélisation porte sur l'arbitrage des conflits entre trains, et plus précisément entre couple de trains. Le problème a été modélisé en un problème de satisfaction de contraintes, appelé CSP. Ce CSP comporte 2 grands types de variables :

- variables d'affectation : parcours (r_i),
- variables d'ordonnancement : dates de départs (st_i) et bornes des intervalles d'incompatibilité ($\underline{t_{ij}}, \overline{t_{ij}}$).

Afin de tenir compte de l'infrastructure et d'éviter tout conflit entre trains, une série de contraintes a été modélisée :

- contraintes d'énumérations entre les parcours et les intervalles d'incompatibilités (cf. section 3.2.2),
- contraintes d'incompatibilités entre les dates de départs de couples de trains (cf. section 3.2.2).

Une série d'améliorations a été apportée à ce modèle, elles permettent de réduire le temps de résolution et l'espace mémoire utilisé, à savoir :

- une contrainte globale permettant d'éviter des solutions symétriques (cf. section 3.3.4),
- une redéfinition de la contrainte d'incompatibilité (cf. section 3.2.2),
- une réduction du nombre de contraintes et de variables (cf. section 3.3.1,

- un ajout de coupes, permettant également d'obtenir une borne inférieure et d'éliminer des solutions symétriques (cf. section 3.3.2),
- une borne supérieure (cf. section 3.3.2).

Le critère adopté pour ce problème est de minimiser la date d'entrée du dernier train dans l'infrastructure. Comme expliqué en section 3.2.3, il est semblable à la maximisation du nombre de trains pouvant circuler dans l'infrastructure en un temps donné.

Les formats de données employées pour représenter des instances de problème ont également été indiqués. Ainsi un ensemble de 18 instances a été expérimenté, dont trois de ces instances proviennent de données réelles du nœud de Pierrefitte-Gonnesse.

Plusieurs variantes de cette modélisation peuvent permettre d'étudier d'autres problèmes ferroviaires tel que le problème de faisabilité et le problème de stabilité d'une grille horaire.

Cette modélisation du problème de capacité d'infrastructure ferroviaire est relativement proche des données réelles. En effet, chaque variable a une représentation physique directe, ainsi une solution donnée pourra plus rapidement être interprétée que si des variables booléennes avaient été utilisées. En contre-partie, comme nous l'avons vu, le critère choisi n'est pas de maximiser la capacité, mais de diminuer le temps d'occupation de l'infrastructure. Ainsi, si le décideur demande l'occupation maximale sur une période donnée, le nombre de trains devra être approximé (grâce notamment à la borne supérieure obtenue), puis augmenté ou diminué en fonction du temps d'occupation obtenu.

Cette modélisation est également extrêmement fine, la discrétisation est à la seconde près. Aussi en terme d'exploitation faire passer un train à la seconde n'est pas gérable. Il s'agit d'une modélisation théorique permettant d'obtenir une solution faisable, mais peut être pas réalisable en pratique. Une extension possible de ce modèle afin de le rendre plus proche de la réalité serait d'ajouter une marge aux intervalles d'incompatibilité entre parcours, afin de prendre en considération les aléas de l'utilisation réelle.

Algorithmes de résolution

La seconde partie de ce travail a porté sur la résolution de ce problème. Les algorithmes de résolution utilisent les techniques de la programmation par contraintes, ainsi un effort important concerne la propagation des événements (affectations, réduction de domaine). C'est pourquoi une partie des améliorations se rapportait à amélioration de la propagation notamment avec la réécriture de la contrainte d'incompatibilité et l'ajout de coupes.

Dans un premier temps, la résolution exacte de ce problème a été étudiée, il en est ressorti que cette résolution n'est pas utilisable pour les tailles de problème à traiter. Aux vues de l'efficacité de l'heuristique de choix de valeurs obtenues, il s'est avéré intéressant d'étudier l'impact d'une

heuristique de recherche à arborescence tronquée. Le choix a porté sur l'heuristique DDS qui permet de tenir compte du nombre de variables déjà affectées dans le processus. Les résultats obtenus sont meilleurs que pour la résolution exacte, mais restent malgré tout trop faibles aux vues de notre objectif. Néanmoins, ses résolutions s'avèrent très efficaces sur des instances de petites tailles, et peuvent ainsi être utilisées dans la résolution approchée, telle que la recherche à voisinage large (LNS).

À partir de l'analyse des résultats obtenus sur la résolution exacte, une résolution approchée était nécessaire. L'algorithme choisi est la métaheuristique à voisinage large LNS, où l'heuristique DDS a été utilisé pour les phases de ré-optimisation et le passage d'un type de voisinage à l'autre a été effectué grâce à la métaheuristique de recherche à voisinage variable VNS.

Les paramètres de cette métaheuristique portent sur la taille et le type de voisinage ainsi que sur la solution initiale. Les résultats des expérimentations réalisées ont montré que le meilleur compromis des paramètres de cette métaheuristique est :

- un voisinage évoluant de manière chronologique sur l'ordre des trains,
- une taille de voisinage augmentant par pas de 2,
- une solution initiale de bonne qualité obtenue par notre borne supérieure.

Les résultats obtenus sont de très bonne qualité et constituent, à notre connaissance, les meilleurs obtenus sur les instances réelles. Néanmoins l'algorithme utilisé peut sûrement encore être amélioré, notamment au niveau des trains se trouvant à l'intérieur de notre solution (les trains compris entre N_{inc} et $N - N_{inc}$), deux pistes pourrait être explorées :

- Implémenter un algorithme permettant de diversifier la recherche, quitte à détériorer la solution.
- Partir d'une solution initiale de nature complètement différente de celle utilisée, notamment en évitant l'utilisation de séquences indentiques qui constituent un « attracteur » dont il est difficile de s'éloigner lors des optimisations des voisinages. Ces solutions initiales pourraient être de moins bonne qualité, mais pourraient permettre de trouver, au final, de meilleures solutions.

Plusieurs remarques sur la résolution et sur les solutions obtenues peuvent être ajoutées :

- L'impact de la solution initiale sur la résolution reste déterminante. Qui plus est l'utilisation de la métaheuristique GRASP (méthode donnant de très bons résultats) n'a pas été très efficace.
- Les solutions obtenues ne comportent que très peu de parcours différents, n'existe-t-il pas un sous-ensemble de parcours efficace ?

Cette dernière question amène la recherche vers d'autres domaines présentés dans la section suivante.

Travail connexe et Perspectives

L'analyse des solutions a mis en évidence que seul un sous-ensemble de parcours était utilisé. Aussi il peut s'avérer intéressant d'examiner les relations entre les parcours et d'utiliser ces relations dans la résolution. Pour ce faire, deux pistes ont été investigues : les techniques permettant d'expliquer les échecs et le data mining.

Programmation par contraintes avec explications

La notion d'explication [Jussien et al.00] permet de déterminer les causes d'un échec (pas de solution possible aux vues des choix déjà effectués) en programmation par contraintes. L'explication d'un échec est obtenue par l'ensemble des contraintes du problème n'étant pas satisfaites. Le but est alors d'arriver à déterminer l'ensemble minimum de contraintes amenant à cette situation, d'en tirer des conclusions et de s'en servir pour la suite de la résolution, comme par exemple en utilisant le concept de nogood.

En ce qui concerne le problème ferrroviaire traité, la structure du problème est constituée par les bornes d'incompatibilités. Autrement dit il existe des paires de parcours qui induisent des écarts minimum de temps plus importants. L'identification de ces paires de valeurs pourraient permettre d'accélérer la preuve d'optimalité. L'idée est d'appliquer un schéma de branchement de type meilleur choix en premier ou Best-First (comme c'est le cas actuellement) suivi du moins bon ou First-Fail (basé sur les paires de parcours les plus incompatibles).

Dans ce contexte, les explications pourraient implicitement revenir sur ces choix contraignants. Nous avons pu tester des algorithmes utilisant des explications sur notre problème, grâce à Hadrien Cambazard (École des Mines de Nantes), l'algorithme testé est l'algorithme DBT : algorithme de back-tracking dynamique [Gingsberg93].

Cet algorithme a été testé sur nos instances ferroviaires. Le tableau 5.27 présente les résultats obtenus sur les instances TGV-MA-CL et TGV-MA. L'algorithme MAC correspond à l'algorithme de résolution proposé par défaut par la bibliothèque CHOCO. L'algorithme DBT correspond à un algorithme de back-tracking dynamique utilisant le système d'explications PALM [Jussien et al.00] (Propagation And Learning with Moves) et utilisant la bibliothèque de contraintes CHOCO. Même si des remises en causes et sauts intelligents s'effectuent (moins de choix effectués), leur nombre et leur gain sont très loin de prendre le dessus sur le temps nécessaire à établir les explications.

Data mining

De manière générale, on peut définir le data mining (d'après Frawley et Piateski-Shapiro [Piateski et al.93] comme l'extraction d'informations ou de connaissances originales, auparavant

instance/nb trains	MAC		DBT	
	temps(s)	nb choix	temps(s)	nb choix
TGV-MA-CL / 3	0	49	0.3	25
TGV-MA-CL / 4	0.1	154	3.4	451
TGV-MA-CL / 5	2.3	28218	136.6	30088
TGV-MA-CL / 6	5.7	106160	163	31920
TGV-MA / 6	2	14478	36.90	13704
TGV-MA / 7	5.7	97676	466.3	126527

TAB. 5.27 – Comparaison des algorithmes MAC et DBT

itinéraire et type de train		bornes d'incompatibilité	
r_i	r_j	$\underline{t_{ij}}$	$\overline{t_{ij}}$
A	A	68	-68
A	B	250	-300
A	C	120	10
B	B	160	-160
B	C	220	-240
C	C	79	-79

TAB. 5.29 – Exemple de parcours dominé

inconnues, potentiellement utiles à partir de gros volumes de données. Selon SAS-INSTITUTE [1], il s'agit du processus de sélection, exploration, modification et modélisation de grandes bases de données afin de découvrir des relations entre les données jusqu'alors inconnues.

Le Data Mining correspond donc à l'ensemble des techniques et des méthodes qui à partir de données permettent d'obtenir des connaissances exploitables. Ces techniques pourraient être utilisées pour notre problème. En effet, lors de la présentation des résultats, il a été montré que très peu de parcours étaient utilisés dans les meilleures solutions. Il en est de même pour les solutions optimales obtenues dans la résolution exacte. Qui plus est, ces parcours sont toujours les mêmes pour un type d'instance donnée, quelque soit sa taille. L'idée serait alors de pouvoir déduire de notre ensemble de parcours initial un sous-ensemble optimal, c'est à dire un sous-ensemble de parcours tel que leur combinaison soit optimale.

Dans ce sens, un travail a déjà effectué afin de pouvoir éliminer les parcours dominés, c'est à dire les parcours ayant l'ensemble de ces intervalles d'incompatibilités supérieurs à l'ensemble des intervalles d'incompatibilité d'au moins un autre. Bien évidemment les comparaisons des intervalles d'incompatibilités s'effectuent toujours par rapport au même parcours, prenons l'exemple suivant :

Exemple Dominance entre parcours

Soit un ensemble de 3 parcours A, B et C, ayant les intervalles d'incompatibilités du tableau 5.29.

Il apparaît ainsi clair que le parcours B est dominé par le parcours A, c'est à dire que les intervalles d'incompatibilités du parcours A sont inclus dans les intervalles d'incompatibilités du parcours B.

De ce fait, les solutions optimales de ce type d'instance ne contiendront pas le parcours B.

Bien que cette étude n'ait pas donné lieu à l'obtention de parcours dominés sur nos instances, elle a néanmoins mis en évidence que certains parcours n'avaient que très peu d'intervalles d'incompatibilités (3 ou 4) moins pénalisants que les autres. Aussi une recherche plus approfondie, tenant compte, par exemple, de la comparaison de plus de deux parcours à la fois, pourrait permettre de trouver des ensembles de parcours dominants et dominés.

Bibliographie

[Backofen et al.99] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In *Principles and Practice of Constraint Programming*, pp. 73–87. 1999.

[Bellaiche97] H. Bellaiche. *Recherche sur la saturation des lignes ferroviaires (rapport d'étape de la phase 1)*. Rapport technique n° 2166/ESF/317-97/RA, France, SYSTRA, mai 1997.

[Benhamou et al.94] F. Benhamou, D. McAllester, and P. Van Hentenryck. Clp(intervals) revisited. In *International Symposium on Logic Programming (ILPS'94)*, pp. 124–138. Ithaca, New York, MIT Press, 1994.

[Bohlin02] M. Bohlin. *Constraint satisfaction by local search*. Rapport technique n° T2002-07, SICS, 2002.

[Bussieck et al.97] M. Bussieck, T. Winter, and U. Zimmermann. Discrete optimization in public rail transport. *Mathematical programming*, 79 : 415–444, 1997.

[Caseau et al.94] Y. Caseau and F. Laburthe. Improving clp scheduling with task intervals. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pp. 369–383. Santa Margherita Ligure, Italy, 1994.

[Cce] D. CCE.

[Choco] CHOCO. Site web de choco, version 1.2.

[Cordeau et al.98] J. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4) : 380–404, 1998.

[Crawford et al.96] J. Crawford, M. L. Ginsberg, E. Luck, and A. Roy. Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, and S. Shapiro (Eds.), *KR'96 : Principles of Knowledge Representation and Reasoning*, pp. 148–159. San Francisco, California, Morgan Kaufmann, 1996.

[Curchod et al.01] A. Curchod and L. Lucchini. *CAPRES : description générale du modèle*. Rapport technique n° 788/5_f, LITEP, juin 2001.

[dCdFU96] U. I. des Chemins de Fer (U.I.C.). *Fiche 405/OR – Liens entre capacité des infrastructures ferroviaires et qualité de l'exploitation*. 1996.

[De givry et al.06] S. De Givry and L. Jeannin. A unified framework for partial and hybrid search methods in constraint programming. *Computers & Operations Research*, 33(10) : 2805–2833, 2006.

[Dechter et al.91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49 : 61–95, 1991.

[Dechter et al.94] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraints satisfaction problems. *Artificial Intelligence*, 68 : 211–241, 1994.

[Dechter90] R. Dechter. Enhancement schemes for constraint processing : backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3) : 273–312, 1990.

[Dechter92] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55(1) : 87–108, 1992.

[Degoutin et al.05a] F. Degoutin and H. Cambazard. A csp model and lns approach for the railway saturation problem. In *Workshop on combination of metaheuristic and local search with Constraint Programming techniques*. Nantes, November 2005.

[Degoutin et al.05b] F. Degoutin and J. Rodriguez. Evaluation of depth-bounded discrepancy on a csp model of the railway saturation problem. In *17th IMACS World Congress Scientific Computation, Applied Mathematics and Simulation*. Paris, France, juillet 2005.

[Degoutin et al.05c] F. Degoutin, J. Rodriguez, and X. Gandibleux. évaluation des performances d'un modèle csp pour le problème de saturation d'infrastructures ferroviaires. In J. Billaut and C. Esswen (Eds.), *ROADEF'05, 6ième congrès de société française de recherche opérationnelle et d'aide à la décision*, pp. 277–294. Presse universitaire François Rabelais, Février 2005.

[Degoutin02] F. Degoutin. *Problèmes bi-objectifs en optimisation combinatoire et capacité d'infrastructures ferroviaires*. Valenciennes, France, Mémoire de DEA, Université de Valenciennes et du Hainaut Cambrésis, 2002.

[Degoutin05] F. Degoutin. évaluation des performances d'un modèle csp pour le problème de saturation d'infrastructures ferroviaires. In *Rencontre avec les doctorants des laboratoires ESTAS, LEOST, LIVIC, LTN.*, pp. 149–166. 2005.

[Delorme et al.01] X. Delorme, X. Gandibleux, and J. Rodriguez. Heuristics for railway infrastructure saturation. In L. Baresi, J.-J. Lévy, R. Mayr, M. Pezzé, G. Taentzer, and C. Zaroliagis (Eds.), *Electronic Notes in Theoretical Computer Science*, pp. 41–55. Elsevier Publisher, 2001.

[Delorme et al.04] X. Delorme, X. Gandibleux, and J. Rodriguez. GRASP for set packing problems. *European Journal of Operational Research*, 159 (2), 2004.

[Delorme03] X. Delorme. *Modélisation et résolution de problèmes liés à l'exploitation d'infrastructures ferroviaires*. Valenciennes, France, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambresis, décembre 2003.

[Ehrgott et al.02] M. Ehrgott and X. Gandibleux. An overview of multiobjective metaheuristics doa. In *Kyoto university*. june 2002.

[Esquirol et al.97] P. Esquirol and P. Lopez. Concepts et outils pour les systemes de production. pp. 133–160. Cepadues Editions, 1997.

[Esquirol et al.01] P. Esquirol, P. Lopez, and M. Huguet. Ordonnancement de la production. pp. 131–167. Hermes Science Publications, 2001.

[Felici et al.92] A. Felici and L. Negri. La qualité de service voyageurs des chemins de fer. *Recherche transports sécurité*, 36 : 47–52, décembre 1992.

[Florio et al.96] L. Florio and L. Mussone. A method of capacity computation for complex railways systems. *World Transport Research , Selected Proceedings of 7th World Conference on Transport Research*, 4 : 275–291, 1996.

[Fo et al.89] T. A. Féo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8 : 67–71, 1989.

[Focacci et al.01a] F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In *MIC'2001 - 4th Metaheuristic International Conference*. Porto Portugal, July 2001.

[Focacci et al.01b] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *CP-01*, pp. 75–92. 2001.

[Fontaine et al.01] M. Fontaine and D. Gauyacq. SISYFE : a toolbox to simulate the railway network functioning for many purposes. some cases of application. In *Proceedings of the World Congress on Railway Research (WCRR 2001)*. 2001.

[Freuder82] E. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1) : 24–32, 1982.

[Fukumori et al.87] K. Fukumori and H. Sano. Fundamental algorithm for train scheduling based on artificial intelligence. *Systems and computer in Japan*, 3(18), 1987.

[Gaschnig77] J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings IJCAI'77*, p. 447. 1977.

[Gensel95] J. Gensel. *Contraintes et représentation de connaissances par objets application au modèle TROPES.* Grenoble, Suisse, Thèse de doctorat, Université Joseph Fournier Laboratoire LIFIA/IMAG, octobre 1995.

[Giger87] P. Giger. Data concept for simulation of railway networks. pp. 67–76, 1987.

[Gingsberg93] M. Gingsberg. Dynamic backtracking. *Artificial Intelligence*, 1 : 25–46, 1993.

[Glover et al.93] F. Glover and M. Laguna. Tabu search. In C. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*. Oxford, England, Blackwell Scientific Publishing, 1993.

[Glover et al.97] F. Glover and M. Laguna. *Tabu Search*. Kluwer academic publishers, Boston, 1997.

[Gondran et al.95] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 1995.

[Group99] E. E. U. Group. *ERTMS/ETCS System Requirements Specification.* – Brussels, UIC, 1999.

[Hachemane97] P. Hachemane. *Évaluation de la capacité de réseau ferroviaire*. Lausanne, Suisse, Thèse de doctorat, École Polytechnique Fédérale de Lausanne, 1997.

[Hansen et al.00] P. Hansen and N. Mladenovic. Handbook of applied optimization. p. 20. février 2000.

[Hansen et al.01] P. Hansen, N. Mladenovic, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4) : 335–350, 2001.

[Hansen et al.03] P. Hansen and N. Mladenovic. A tutorial on variable neighborhood search. *Les cahiers du GERARD*, 2003.

[Haralick et al.80] R. M. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 : 263–313, 1980.

[Harvey et al.95] W. Harvey and M. Ginsberg. Limited discrepancy search. In C. S. Mellish (Ed.), *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95) ; Vol. 1*, pp. 607–615. Montréal, Québec, Canada, Morgan Kaufmann, 1995, August 20-25 1995.

[Ilog99] ILOG. *CPLEX (user's manual)*, 1999.

[Ilog01] ILOG. *ILOG Concert Technology 1.0 (user's manual)*, 2001.

[Ilog02] ILOG. *ILOG Solver 5.3 (user's manual)*, 2002.

[Jussien et al.00] N. Jussien and V. Barichard. The palm system : explanation-based constraint programming. In *TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pp. 118–133. 2000.

[Kirkpatrick et al.83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598 : 671–680, 1983.

[Korf96] R. E. Korf. Improved limited discrepancy search. In *AAAI/IAAI*, pp. 286–291. 1996.

[Labouisse et al.01] V. Labouisse and H. Djellab. DEMIURGE : a tool for the optimisation and the capacity assessment for railway infrastructure. In *Proceedings of the World Congress on Railway Research (WCRR 2001)*. 2001.

[Labouisse et al.02] V. Labouisse and H. Djellab. DEMIURGE : un outil d'optimisation et d'évaluation de la capacité d'infrastructure ferroviaire. 4ème congrès de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2002), Paris, France, 20-22 février 2002.

[Lhomme93] O. Lhomme. Consistency techniques for numeric csps. In *International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 232–238. 1993.

[Mackworth77] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 : 99–118, 1977.

[Marliere01] G. Marlière. *Développement d'une interface graphique pour la gestion des circulations ferroviaires*. Rapport technique n° RE-01-708-FR, INRETS, juin 2001.

[Mascis et al.04] A. Mascis, D. Paciarelli, and M. Pranzo. Scheduling model for short-term railway traffic regulation. In L. N. in Economics and M. Systems (Eds.), *9th International Conference on Computer-Aided Scheduling of Public Transport*. San Diego CA (U.S.A.), août 2004.

[Meseguer et al.01] P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artificial Intelligence*, 129(1–2) : 133–163, 2001.

[Mladenovic et al.97] N. Mladenović and P. Hansen. Variable neighborhood search. *Comps. in Opns. Res.*, 24 : 1097–1100, 1997.

[Montanari74] Montanari. Fundamental properties and applications to picture processing. *Information Sciences*, 66(7) : 95–132, 1974.

[Osman et al.96] I. H. O. Osman and G. Laporte. Meta-heuristics : a bibliography. *Annals of Operations Research*, 63 : 513–628, 1996.

[Pachl04] J. Pachl. Railway operation and control. Gorham Printing, Decembre 2004.

[Paciarelli et al.02] D. Paciarelli and P. Brucker. *Traffic regulation and co-operation methodolgies*. Rapport technique n° D3, Université de Genève, mars 2002.

[Paciarelli et al.06] D. Paciarelli and M. Pranzo. Speed regulation in rail network. In I. Symposium (Ed.), *CTS*. Delft (The Netherlands), août 2006.

[Pesant et al.96] G. Pesant and M. Gendreau. A view of local search in constraint programming. In *Principles and Practice of Constraint Programming - CP96 : Proceedings of the Second International Conference*, pp. 353–366. volume 1118 of Lecture Notes in Computer Science, 1996.

[Pesant et al.99] G. Pesant and M. Gendreau. A constraint programming framework for local search methods. In *Journal of Heuristics 5*, pp. 255–279. Kluwer Academic Publishers, 1999.

[Piateski et al.93] G. Piateski, G. Piatetski-Shapiro, and W. Frawley. *Knowledge Discovery in Databases brings together current research on the exciting problem of discovering useful and interesting knowledge in databases*. AAAi Press / MIT Press, Menlo Park CA, 1993.

[Pirlot02] M. Pirlot. Métaheuristiques pour l'optimisation combinatoire : un aperçu général. In P. M. TEGHEM J. (Ed.), *Optimisation approchée en recherche opérationnelle - Recherches locales, réseaux neuronaux et satisfaction de contraintes*, pp. 25–55. Paris, Hermès Science Publications, 2002.

[Pitsoulis et al.02] L. Pitsoulis and M. Mauricio G.C. Resende. Greedy randomized adaptive search procedures. In P. Pardalos and M. Resende (Eds.), *Handbook of Applied Optimization*, pp. 168–183. Oxford University Press, New York, 2002.

[Prcovic02] N. Prcovic. Quelques variantes de LDS. In *VIII^e Journées Nationales sur les Problèmes NP-Complets (JNPC'02)*, pp. 195–208. Nice, 2002.

[Puget03] J. F. Puget. Symetry breaking using stabilizers. In *CP'03*. 2003.

[Puget05] J. F. Puget. Symmetry breaking revisited. *Constraints*, 10(1) : 23–46, 2005.

[Purdom83] P. W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21 : 117–133, 1983.

[Reeves95] C. Reeves. *Modern heuristic techniques for combinatorial problems*. London, McGraw-Hil, 1995.

[Resende et al.02] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger (Eds.), *Handbook in Metaheuristics*, pp. 219–249. Kluwer academic publishers, Boston, 2002.

[Rivier et al.01] R. Rivier, L. Lucchini, and A. Curchod. Analysing the capacity of railway networks : Summing up the experience. In 9^th *World Conference on Transportation Research*. July 2001.

[Rodriguez02] J. Rodriguez. *Analyse des conflits de circulations ferroviaires.* Rapport technique n° RR-02-734-FR, INRETS-ESTAS, 2002.

[Rousseau et al.99] L. Rousseau, M. Gendreau, and G. Pesant. Une hybridation de programmation par contraintes et recherche locale pour le problème de tournées de véhicule. In *JFPLC'99. Journée francophone pour la programmation logique par contrainte*, pp. 145–160. Lyon, Hermès Ed., 1999.

[Schiex et al.94] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *International Journal on Artificial Intelligence Tools (IJAIT)*, 3(2) : 187–207, 1994.

[Schneider97] F. Schneider. *Recherche sur la saturation des lignes ferroviaires (rapport d'étude de la phase 2).* Rapport technique n° 2166/ESF/721-97/RA, France, SYSTRA, décembre 1997.

[Schrijver et al.94] A. Schrijver and A. Steenbeek. *Dienstregeling ontwikkeling voor Railned (Timetable development for Railned).* Rapport technique, Amsterdam, Netherlands, C.W.I., 1994. Cadans 1.0.

[Schwalb et al.97] E. Schwalb and R. Dechter. Processing disjunctions in temporal constraint networks. *Artificial intelligence*, 93 : 29–61, 1997.

[Shaw98] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science*, 1520 : 417–431, 1998.

[Van den berg et al.94] J. Van den Berg and M. Odijk. DONS : computer aided design of regular service timetables. In T. K. S. Murphy, B. Mellitt, C. A. Brebbia, G. Sciutto, and S. Sone (Eds.), *Computers in Railway IV, proceedings of the Fourth International Conference on Computer Aided Design, Manufacture and Operation in the Railway and Other Advanced Mass Transit Systems (COMPRAIL 94)*, pp. 109–115. Computational Mechanics Publications, Southampton Boston, 1994.

[Walsh97] T. Walsh. Depth-bounded discrepancy search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1388–1393. Août 1997.

[Zwaneveld et al.96] P. J. Zwaneveld, L. G. Kroon, H. E. Romeijn, M. Salomon, S. Dauzère-Pérès, S. P. Van Hoesel, and H. W. Ambergen. Routing trains through railway stations : Model formulation and algorithms. *Transportation Science*, 30(3) : 181–194, août 1996.

[Zwaneveld97] P. J. Zwaneveld. *Railway planning - routing of trains and allocation of passenger lines.* Rotterdam, Netherlands, Thèse de doctorat, Rotterdam school of management, TRAIL research school, 1997.

Annexes

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
8 trains	*r-t-st*	0.43s	0.45s	0.44s	0.4s	0.46s
	t-r-st	2.55s	2.5s	2.54s	2.48s	2.51s
9 trains	*r-t-st*	0.97s	0.9s	0.98s	0.92s	0.95s
	t-r-st	19.2s	18s	19.8s	17.7s	18.8s

TAB. 5.30 – Comparaison des ordres de choix de variables (tableau 5.2 page 123)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
8 trains	*r-t-st*	0.21s	0.5s	0.43s	0.43s	0.18s
	t-r-st	8.7s	13.3s	12.1s	11.1s	7.5s
9 trains	*r-t-st*	0.97s	1.9s	1.7s	1.3s	0.58s
	t-r-st	523.23s	612.3s	601.43s	550.3s	524.29s

TAB. 5.31 – Comparaison des ordres de choix de variables (tableau 5.2 page 123)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
8 trains	*r-t-st*	0.97s	1.9s	1.8s	1.6s	1.23s
	t-r-st	550.2s	712.8s	689.1s	682.6s	650.65s
9 trains	*r-t-st*	2.4s	5.1s	4.3s	4s	3.2s
	t-r-st	1308.3s	2109.2s	1912.7s	1803.7s	1546.6s

TAB. 5.32 – Comparaison des ordres de choix de variables (tableau 5.2 page 123)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
# nœuds	M_{12345} (12 trains)	3560	4059	3979	3755	3547
	M_{2345} (12 trains)	3563	4065	3981	3770	3541
temps	M_{12345} (12 trains)	1.9 s	2.56 s	2.34 s	2.01 s	1.84 s
	M_{2345} (12 trains)	1.9 s	2.5 s	2.32 s	1.99	1.84 s
toutes tailles :	# nœuds	0%	0%	0%	0%	0%
M_{2345} / M_{12345}	temps	1.5%	2%	2%	2%	2%

TAB. 5.33 – Impact de la contrainte NGBS (tableau 5.3 page 124)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
# nœuds	M_{12345} (12 trains)	5198	7373	7265	7165	6619
	M_{2345} (12 trains)	5201	7383	7269	7171	6621
temps	M_{12345} (12 trains)	5.32 s	8.45 s	7.8 s	7.71 s	7.37 s
	M_{2345} (12 trains)	5.3 s	8.44 s	7.77 s	7.65 s	7.14 s
toutes tailles :	# nœuds	0%	0%	0%	-0.5%	0%
M_{2345} / M_{12345}	temps	1%	2%	3%	4%	4%

TAB. 5.34 – Impact de la contrainte NGBS (tableau 5.3 page 124)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
# nœuds	M_{12345} (12 trains)	6503	7320	7120	6901	6641
	M_{2345} (12 trains)	6503	7320	7120	6901	6641
temps	M_{12345} (12 trains)	11.4s	20.4s	18.2s	18s	15.85s
	M_{2345} (12 trains)	11.2s	20.2s	18.3s	17.9s	16s
toutes tailles :	# nœuds	-1%	0%	-1%	-1%	-1%
M_{2345} / M_{12345}	temps	0%	0%	0%	0%	0%

TAB. 5.35 – Impact de la contrainte NGBS (tableau 5.3 page 124)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
temps de	M_{245}	150 s	205 s	192 s	191 s	162 s
résolution	M_{2345}	140 s	190 s	181 s	178 s	151 s
valeur de N^*_{inc}		15	16	15	15	15
réduction du nombre de contraintes		86%	85%	85%	85%	84%

TAB. 5.36 – Impact de l'élimination des contraintes et variables inutiles sur des instances à 100 trains (tableau 5.5 page 126)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
temps de résolution	M_{245}	190 s	310 s	272 s	253 s	220 s
	M_{2345}	182 s	287 s	257 s	235 s	204 s
valeur de N^*_{inc}		18	18	19	18	17
réduction du nombre de contraintes		82%	83%	81%	82%	82%

TAB. 5.37 – Impact de l'élimination des contraintes et variables inutiles sur des instances à 100 trains (tableau 5.5 page 126)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
temps de résolution	M_{245}	297 s	412 s	398 s	358 s	315 s
	M_{2345}	271 s	351 s	341 s	309 s	278 s
valeur de N^*_{inc}		21	21	21	21	20
réduction du nombre de contraintes		77%	79%	81%	78%	80%

TAB. 5.38 – Impact de l'élimination des contraintes et variables inutiles sur des instances à 100 trains (tableau 5.5 page 126)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
12 trains	M_{235}	4.5 s	6.9 s	6.7 s	4.6 s	3.9 s
	M_{2345}	1.9 s	2.5 s	2.32 s	1.99	1.84 s
moyenne M_{235}/M_{2345}		83%	84%	85%	84%	83%

TAB. 5.39 – Impacts des coupes sur la résolution (tableau 5.6 page 127)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
12 trains	M_{235}	14.3 s	17.9 s	16.9 s	16.1 s	15.05 s
	M_{2345}	5.3 s	8.44 s	7.77 s	7.65 s	7.14 s
moyenne M_{235}/M_{2345}		87%	88%	88%	88%	87%

TAB. 5.40 – Impacts des coupes sur la résolution (tableau 5.6 page 127)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
12 trains	M_{235}	54.21 s	75.3 s	73.6 s	59.5 s	55.54 s
	M_{2345}	11.2s	20.2s	18.3s	17.9s	16s
moyenne M_{235}/M_{2345}		82%	81%	82%	85%	80%

TAB. 5.41 – Impacts des coupes sur la résolution (tableau 5.6 page 127)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
12 trains	M_{234}	2.9 s	4.1 s	3.9 s	3.8 s	3.65 s
	M_{2345}	1.9 s	2.5 s	2.32 s	1.99	1.84 s
moyenne M_{234}/M_{2345}		28%	27%	29%	31%	25%

TAB. 5.42 – Impacts de la borne sur la résolution (tableau 5.7 page 127)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
12 trains	M_{234}	7.6 s	10.8 s	10.2 s	9.3 s	8.65 s
	M_{2345}	5.3 s	8.44 s	7.77 s	7.65 s	7.14 s
moyenne M_{234}/M_{2345}		23%	21%	23%	17%	19%

TAB. 5.43 – Impacts de la borne sur la résolution (tableau 5.7 page 127)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
12 trains	M_{234}	12.4 s	22.6 s	20.6 s	19.3 s	17.7 s
	M_{2345}	11.2s	20.2s	18.3s	17.9s	16s
moyenne M_{234}/M_{2345}		35%	46%	40%	39%	40%

TAB. 5.44 – Impacts de la borne sur la résolution (tableau 5.7 page 127)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
12 trains	M_{345}	4.6 s	6.42 s	6.31 s	5.89 s	5.93 s
	M_{2345}	1.9 s	2.5 s	2.32 s	1.99	1.84 s
moyenne M_{345}/M_{2345}		58%	59%	60%	56%	57%

TAB. 5.45 – Comparaison des deux implémentations des contraintes d'incompatibilités (tableau 5.8 page 129)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
12 trains	M_{345}	12.4 s	19.4 s	19.12 s	16.17 s	17.36 s
	M_{2345}	5.3 s	8.44 s	7.77 s	7.65 s	7.14 s
moyenne M_{345}/M_{2345}		68%	67%	67%	64%	69%

TAB. 5.46 – Comparaison des deux implémentations des contraintes d'incompatibilités (tableau 5.8 page 129)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
12 trains	M_{345}	29.4 s	41.2 s	39.4 s	36.1 s	32.9 s
	M_{2345}	11.2s	20.2s	18.3s	17.9s	16s
moyenne M_{345}/M_{2345}		58%	61%	60%	61%	60%

TAB. 5.47 – Comparaison des deux implémentations des contraintes d'incompatibilités (tableau 5.8 page 129)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
st_N	4590s	4662s	5464s	3450s	5417s
LNS+VNS	100%	100%	100%	100%	100%

TAB. 5.48 – Meilleures valeurs (tableau 5.25 page 144)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
st_N	4026s	4437s	4950s	4410s	3991s
LNS+VNS	100%	0%	100%	100%	100%

TAB. 5.49 – Meilleures valeurs (tableau 5.25 page 144)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
st_N	2970s	3225s	4264s	3185s	3807s
LNS+VNS	100%	100%	100%	0%	0%

TAB. 5.50 – Meilleures valeurs (tableau 5.25 page 144)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
temps DDS/exact	8 trains	-32%	-37%	-36%	-36%	-34%
	12 trains	-35%	-42%	-42%	-40%	-41%
qualité solution DDS/exact	8 trains	0.95%	0.88%	0.88%	0.9%	0.89%
	12 trains	0.91%	0.9%	0.9%	0.9%	0.89%

TAB. 5.51 – Temps de résolution et qualité de résolution pour des tailles d'instances différentes(tableau 5.10 page 133)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
temps DDS/exact	8 trains	-28%	-34%	-33%	-33%	-32%
	12 trains	-32%	-38%	-39%	-39%	-37%
qualité solution DDS/exact	8 trains	0%	0.8%	1%	0.9%	0.8%
	12 trains	0%	0.9%	1%	0.9%	0.7%

TAB. 5.52 – Temps de résolution et qualité de résolution pour des tailles d'instances différentes(tableau 5.10 page 133)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
temps DDS/exact	8 trains	-27%	-32%	-34%	-31%	-31%
	12 trains	-30%	-39%	-41%	-36%	-34%
qualité solution DDS/exact	8 trains	0.9%	0.8%	0.5%	0.8%	1%
	12 trains	1%	0.8%	0.5%	0.8%	0.9%

TAB. 5.53 – Temps de résolution et qualité de résolution pour des tailles d'instances différentes (tableau 5.10 page 133)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
exacte / DDS	0.6%	0.8%	0.8%	0.8%	0.75%

TAB. 5.54 – Comparaison des valeurs de solutions obtenues en utilisant DDS ou une résolution exacte (tableau 5.14 page 137)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
exacte / DDS	0.7%	0.9%	0.9%	0.8%	0.7%

TAB. 5.55 – Comparaison des valeurs de solutions obtenues en utilisant DDS ou une résolution exacte (tableau 5.14 page 137)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
exacte / DDS	0.6%	1%	0.9%	1%	1%

TAB. 5.56 – Comparaison des valeurs de solutions obtenues en utilisant DDS ou une résolution exacte (tableau 5.14 page 137)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
Δ_1 / Δ_2	1%	2%	0.5%	2%	2%
Δ_1 / Δ_3	-1%	-1.5%	-1.5%	-0.5%	-0.5%

TAB. 5.57 – Ratio entre les solutions obtenues selon le pas d'augmentation du voisinage (tableau 5.15 page 138)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
Δ_1 / Δ_2	1.5%	2%	2.5%	2%	2%
Δ_1 / Δ_3	0%	1%	0%	0.5%	1%

TAB. 5.58 – Ratio entre les solutions obtenues selon le pas d'augmentation du voisinage (tableau 5.15 page 138)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
Δ_1 / Δ_2	1%	1.5%	2%	1.5%	1.5%
Δ_1 / Δ_3	-1%	-2%	-1.5%	-2%	-1%

TAB. 5.59 – Ratio entre les solutions obtenues selon le pas d'augmentation du voisinage (tableau 5.15 page 138)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
chrono / aléa	-2.5%	-3%	-3%	-3.5%	-3%

TAB. 5.60 – Ratio entre les solutions obtenues avec les différents types de voisinages (tableau 5.16 page 138)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
chrono / aléa	-6%	-4%	-5%	-5%	-5%

TAB. 5.61 – Ratio entre les solutions obtenues avec les différents types de voisinages (tableau 5.16 page 138)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
chrono / aléa	-3%	-5%	-3%	-4%	-5%

TAB. 5.62 – Ratio entre les solutions obtenues avec les différents types de voisinages (tableau 5.16 page 138)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
chrono	4590s	4662s	5521s	3450s	5417s
aléa	4590s	4662s	5464s	3432s	5422s

TAB. 5.63 – Meilleures valeurs selon le type de voisinage (tableau 5.17 page 139)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
chrono	4026s	4437s	4950s	4410s	3991s
aléa	3960s	4443s	4950s	4353s	3991s

TAB. 5.64 – Meilleures valeurs selon le type de voisinage (tableau 5.17 page 139)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
chrono	2970s	3261s	4264s	3185s	3807s
aléa	2970s	3225s	4264s	3185s	3839s

TAB. 5.65 – Meilleures valeurs selon le type de voisinage (tableau 5.17 page 139)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
(LNS+VNS / LNS) séquence	0%	-1%	-0.5%	-0.5%	-0.5%
(LNS+VNS / LNS) MQ	1%	4%	2%	4%	4%

TAB. 5.66 – Ratio entre les solutions obtenues par LNS et LNS+VNS (tableau 5.18 page 139)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
(LNS+VNS / LNS) séquence	0%	0%	1%	1.5%	0%
(LNS+VNS / LNS) MQ	3%	6%	7%	4%	5%

TAB. 5.67 – Ratio entre les solutions obtenues par LNS et LNS+VNS (tableau 5.18 page 139)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
(LNS+VNS / LNS) séquence	0%	-0.5%	0%	0%	0.5%
(LNS+VNS / LNS) MQ	3%	5%	5%	4%	3%

TAB. 5.68 – Ratio entre les solutions obtenues par LNS et LNS+VNS (tableau 5.18 page 139)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
séquence	LNS+VNS	14	11	12	10	13
	LNS	14	11	12	10	13
MQ	LNS+VNS	9	6	8	6	6
	LNS	13	8	11	9	9

TAB. 5.69 – Taille moyenne des voisinages à la fin de la résolution (en nombre de parcours relachées)(tableau 5.19 page 141)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
séquence	LNS+VNS	13	10	9	11	12
	LNS	13	10	9	11	12
MQ	LNS+VNS	11	8	7	9	10
	LNS	12	9	7	11	11

TAB. 5.70 – Taille moyenne des voisinages à la fin de la résolution (en nombre de parcours relachées)(tableau 5.19 page 141)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
séquence	LNS+VNS	15	8	9	10	8
	LNS	16	10	10	10	9
MQ	LNS+VNS	8	4	5	8	5
	LNS	10	7	7	9	7

TAB. 5.71 – Taille moyenne des voisinages à la fin de la résolution (en nombre de parcours relachées)(tableau 5.19 page 141)

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
séquence/MQ	LNS+VNS	-10%	-17%	-16%	-12%	-10%
	LNS	-12%	-16%	-17%	-12%	-13%

TAB. 5.72 – Ratio entre les résolutions utilisant une bonne et mauvaise solution initiale (tableau 5.21 page 141)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
séquence/MQ	LNS+VNS	-14%	-16%	-17%	-14%	-14%
	LNS	-10%	-12%	-12%	-11%	-10%

TAB. 5.73 – Ratio entre les résolutions utilisant une bonne et mauvaise solution initiale (tableau 5.21 page 141)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
séquence/MQ	LNS+VNS	-63%	-51%	-15%	-19%	-37%
	LNS	-63%	-49%	-9%	-10%	-34%

TAB. 5.74 – Ratio entre les résolutions utilisant une bonne et mauvaise solution initiale (tableau 5.21 page 141)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
diff séquence	8%	7%	8%	5%	7%
diff MQ	85%	81%	88%	90%	76%

TAB. 5.75 – Écart entre les solutions initiales et finales (tableau 5.22 page 142)

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
diff séquence	9%	10%	6%	6%	9%
diff MQ	91%	79%	88%	87%	90%

TAB. 5.76 – Écart entre les solutions initiales et finales (tableau 5.22 page 142)

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
diff séquence	12%	11%	7%	9%	11%
diff MQ	92%	91%	85%	80%	87%

TAB. 5.77 – Écart entre les solutions initiales et finales (tableau 5.22 page 142)

type d'instances	aléa1				
numéro d'instances	1	2	3	4	5
possibles	19	19	19	19	19
solutions initiales	1	2	2	3	2
solutions finales	4	5	6	5	5

TAB. 5.78 – Comparaison du nombre de parcours possibles, des solutions initiales et ceux retenus dans les solutions finales (tableau 5.23 page 142).

type d'instances	aléa2				
numéro d'instances	1	2	3	4	5
possibles	26	26	26	26	26
solutions initiales	2	2	3	4	4
solutions finales	6	7	5	5	7

TAB. 5.79 – Comparaison du nombre de parcours possibles, des solutions initiales et ceux retenus dans les solutions finales (tableau 5.23 page 142).

type d'instances	aléa3				
numéro d'instances	1	2	3	4	5
possibles	37	37	37	37	37
solutions initiales	1	4	3	3	4
solutions finales	6	3	5	5	6

TAB. 5.80 – Comparaison du nombre de parcours possibles, des solutions initiales et ceux retenus dans les solutions finales (tableau 5.23 page 142).

type d'instances		aléa1				
numéro d'instances		1	2	3	4	5
LNS+VNS / LNS+VNS+SE	diff nb parcours	0	0	0	0	0
	qualité solution	0%	-0.5%	-1%	-0.5%	-0.5%

TAB. 5.81 – Impact de la recherche de solution équivalentes (tableau 5.24 page 144)

type d'instances		aléa2				
numéro d'instances		1	2	3	4	5
LNS+VNS / LNS+VNS+SE	diff nb parcours	0	0	0	0	0
	qualité solution	-2%	-2%	-2%	-0.5%	-1%

TAB. 5.82 – Impact de la recherche de solution équivalentes (tableau 5.24 page 144)

type d'instances		aléa3				
numéro d'instances		1	2	3	4	5
LNS+VNS / LNS+VNS+SE	diff nb parcours	0	2	0	1	2
	qualité solution	-5%	1%	-4%	-3%	-4%

TAB. 5.83 – Impact de la recherche de solution équivalentes (tableau 5.24 page 144)

Résumé de la modélisation

Références pour les notations et contraintes utilisées :

Variables et constantes employées

- T : l'ensemble des trains,
- $N = card(T)$: le nombre de trains,
- Z : l'ensemble des zones de l'infrastructure. Une zone est la plus petite entité permettant la détection de la présence des trains sur la voie. Le dispositif le plus utilisé pour obtenir cette information est un dispositif électrique appelé "circuit de voie" (c.f. figure 1.4, section 1.1)
- It : l'ensemble des itinéraires de l'infrastructure,
- G : l'ensemble des types de trains possibles (TGV, trains corails, trains de marchandise),
- R : l'ensemble des itinéraires associé au type de trains ($R = G \times It$),
- **les variables "parcours"** $\boxed{r_i}$, avec $dom(r_i) \subset R$, $i \in T$. Ces variables représentent l'itinéraire et le type du train i. R inclut donc l'ensemble des combinaisons d'itinéraires et de type de trains possibles pour le trains i.
- **les variables "bornes d'incompatibilités"** $\boxed{\underline{t_{ij}^k}, \overline{t_{ij}^k}}$, avec $dom(\underline{t_{ij}^k}) \subset \underline{I}$ et $dom(\overline{t_{ij}^k}) \subset \overline{I}$, $i \in T, k = 1, \ldots, card(K^{r_i,r_j})$ Ces variables représentent les valeurs des bornes des $card(K^{r_i,r_j})$ intervalles d'incompatibilités entre le couple de trains (i,j). Ces variables permettent de déterminer les écarts minimaux entre les dates d'entrées des trains dans l'infrastructure afin d'éviter les conflits.
- **les variables "dates d'entrées"** $\boxed{st_i}$, avec $i \in T$. Le pas de discrétisation choisi est la seconde.

Modèle complet M_0

Les références des équations et des pages où ces équations ont été détaillées est indiqué.

$$
\left[\begin{array}{lll}
\min(st_N) & & cf.\ 3.12\ page\ 82 \\
s/c & & \\
\quad \textit{enum}(r_i, r_j, \underline{t_{ij}}, \overline{t_{ij}}), & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}} & cf.\ 3.4\ page\ 78 \\
\quad \delta_{ij} \notin]\underline{t_{ij}}, \overline{t_{ij}}[, & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}} \ //inc_2 & \\
\quad (r_i = a) \wedge (r_j = b) \rightarrow \delta_{ij} \notin]\underline{t^k_{ab}}, \overline{t^k_{ab}}[, & \forall i,j \in T^2, i<j, j<i+N^*_{bt_{inc}}, & \\
\qquad \forall k = 2, \ldots, card(K^{a,b}) & & cf.\ 3.22\ page\ 89 \\
\quad st_{i+s} \geq st_i + m^*_s, & \forall i \leq N-s, s = 2, \ldots, N_s & cf.\ 3.16\ page\ 87 \\
\quad st_N \leq st^*_{n_{seq}+1} \times \lfloor N/n_{seq} \rfloor + st\_seq^*_{N \bmod n_{seq}} & & cf.\ 3.19\ page\ 88 \\
\quad NGBS(r, st), & \forall r_i, st_i | st_i \textit{fixé}, \exists j | st_j = st_{j+1} & cf.\ 3.22\ page\ 91
\end{array}\right. \quad (5.1)
$$

La contrainte inc_1 (3.6 page 80) également utilisée dans le mémoire :

$$
\begin{aligned}
((st_i - st_j) \leq \underline{t^1_{ij}} \vee (st_i - st_j) \geq \overline{t^1_{ij}}), \forall i,j \in T^2, \\
(r_i = a) \wedge (r_j = b) \rightarrow ((st_i - st_j) \leq \underline{t^k_{ab}} \vee (st_i - st_j) \geq \overline{t^k_{ab}}), \\
\forall i,j \in T^2,\ \forall k = 2, \ldots, card(K^{a,b}),
\end{aligned}
$$

avec $\underline{t^k_{ab}}$ la valeur de la borne d'incompatibilité sur le $k^{ième}$ intervalle lorsque $r_i = a$ et $r_j = b$, de même pour $\overline{t^k_{ab}}$.

Résumé :

L'objectif de cette thèse est la résolution du problème d'évaluation de capacité d'infrastructures ferroviaires au niveau d'un nœud ou d'une gare. Le transport ferroviaire se place dans un domaine très concurentiel, aussi malgré ses atouts (développement durable notamment) il se doit d'améliorer son efficacité et la qualité des services aux clients. Pour ce faire un travail conséquent de planification des ressources est effectué. Le problème d'évaluation de la capacité d'infrastructures ferroviaires permet de situer les limites du réseau et d'étudier l'impact de toute modification d'infrastructure.

La modélisation proposée consiste en un problème de satisfaction de contraintes (CSP). La résolution exacte de ce problème par les bibliothèques d'ILOG SOLVER montre des limites, aussi des améliorations sont proposées tant au niveau du modèle que de la résolution. Les améliorations portent sur la réduction du nombre de contraintes, la ré-écriture des contraintes temporelles, la suppression de symétries, des coupes et une borne supérieure. Un algorithme de résolution approchée est proposé, il utilise les principes de recherche à voisinage large (LNS) et variables (VNS) ainsi que de la recherche à arborescence tronquée (DDS).

Les différents algorithmes et améliorations sont validés sur des instances réelles provenant du nœud ferroviare de Pierrefite-Gonesse ainsi que sur un ensemble d'instance aléatoires. Les résultats obtenus sont les meilleurs connus à ce jour.

Discipline : Informatique

Mots clés : Capacité d'infrastructure ferroviaire, Modélisation, Programmation par Contraintes, Métaheuristiques, LNS

Laboratoires : INRETS-ESTAS, 20 rue Élisée Reclus, F-59650, Villeneuve d'Ascq, France
LAMIH-UMR CNRS 8530, UVHC, "Le Mont Houy", F-59313, Valenciennes Cedex 9, France