Recherche locale haute performance pour la planification des interventions à France Télécom

Bertrand Estellon[1] **Frédéric Gardi**[2] **Karim Nouioua**[1]

[1] Laboratoire d'Informatique Fondamentale, Faculté des Sciences de Luminy, Marseille

[2] Bouygues e-lab, Paris

{estellon,nouioua}@lif.univ-mrs.fr fgardi@bouygues.com

Résumé

Dans cette note, nous présentons les travaux de recherche et développement que nous avons réalisés dans le cadre du Challenge ROADEF 2007. Le sujet de la compétition, proposé par France Télécom, consiste en la planification des interventions de maintenance et des équipes de techniciens nécessaires à leurs réalisations. Le problème considéré est un problème d'optimisation combinatoire difficile à plusieurs titres : il contient plusieurs sous-problèmes NP-difficiles et son échelle (des centaines d'interventions et de techniciens) induit une combinatoire gigantesque. Nous décrivons une heuristique à base de recherche locale permettant de résoudre ce problème de manière effective et efficace ; cet algorithme a été classé 2ème du challenge toutes catégories confondues (sur les 35 équipes ayant soumis leurs travaux). En outre, de ce travail transparaît une méthodologie pour la mise en oeuvre d'heuristiques à base de recherche locale hautement performantes.

Abstract

In this note, we present the works of research and development realized in the context of ROADEF 2007 Challenge. The subject of the contest, proposed by France Télécom, consists in planning maintenance interventions and teams of technicians needed for their achievements. The problem considered is a hard combinatorial optimization problem : it contains several NP-hard subproblems and its scale (some hundreds of interventions and technicians) induces a huge combinatorics. We describe an effective and efficient local search-based heuristic to solve this problem ; this algorithm was ranked 2nd of the competition (over the 35 teams who have submitted their works). In addition, this work reveals a methodology for engineering high-performance local search heuristics.

1 Introduction

1.1 Présentation du problème

De manière générale, le problème proposé par France Télécom comme sujet du Challenge ROADEF 2007 [3] (compétition organisée tous les deux ans par la Société Française de Recherche Opérationnelle et d'Aide à la Décision) peut être vu comme un problème d'ordonnancement de tâches avec affectation de ressources. Ici, les tâches à planifier sont des interventions et leur réalisation nécessite l'affectation de ressources humaines, des techniciens, possédant un certain niveau de compétence dans différents domaines. Les interventions sont plus ou moins prioritaires ; au total, 4 priorités sont définies. L'objectif est alors de minimiser un coût fonction des dates de fin des interventions les plus tardives pour chacune des priorités.

Formellement, l'entrée du problème est composée de n interventions I_i et de m techniciens T_t. À chaque technicien T_t est associé son niveau de compétence $C(t,d)$ dans le domaine d et sa présence $P(t,j)$ le jour j (1 si présent, 0 sinon). De même, chaque intervention I_i possède plusieurs caractéristiques : $D(i)$ sa durée d'exécution, $R(i,d,l)$ le nombre de techniciens de niveau l dans le domaine d nécessaire à son exécution, $Z(i)$ son niveau de priorité.

À propos des compétences, il faut noter que les différents domaines de compétence sont disjoints mais que les niveaux à l'intérieur de chaque domaine sont hiérarchiques. Ainsi, un technicien de niveau l dans le domaine d peut effectuer toute intervention nécessitant un niveau de compétence inférieur ($l' < l$) dans le même domaine. En conséquence, les constantes $R(i,d,l)$ sont cumulatives, c'est-à-dire qu'elles spéci-

fient le nombre nécessaire de techniciens de niveau au moins l dans le domaine d. Par exemple, pour une intervention I_i qui requiert pour le domaine d deux technicien de niveau 1 et un technicien de niveau 3, nous aurons $R(i,d,0) = 3$, $R(i,d,1) = 3$, $R(i,d,2) = 1$, $R(i,d,3) = 1$ et $R(i,d,l) = 0$ pour tout $l \geq 4$. Les $R(i,d,l)$ ainsi définis sont donc croissants selon l'indice l : $R(i,d,l) \leq R(i,d,l')$ pour tout $l \leq l'$.

De là, la notion d'équipe de techniciens apparaît. En effet, les techniciens (présents) doivent chaque jour être regroupés en équipes (même si une équipe peut n'être composée que d'un seul technicien). Nous insistons sur le fait qu'une équipe est constituée pour la journée entière (pour des raisons pratiques). Ainsi, le problème consiste à constituer journellement des ensembles de techniciens, les équipes, et à leur affecter des ensembles d'interventions de façon à minimiser une fonction de coût dépendant des dates de fin des interventions. Deux contraintes portent sur cette affectation : la somme des durées des interventions (exécutées de façon séquentielle) ne peut excéder la durée d'une journée de travail fixée à $H = 120$ et la somme des compétences de l'équipe doit être en adéquation avec les compétences requises par l'ensemble des tâches dans chacun des domaines. En définitive, une solution du problème est représentée comme suit : pour chaque jour j, l'équipe $E_{j,e}$ à laquelle appartient le technicien T_t (l'équipe $E_{j,0}$ contiendra tous les techniciens non présents ce jour) ; pour chaque intervention I_i, le jour j_i et l'heure de début h_i de son exécution, ainsi que l'équipe $E_{j,e}$ chargée de cette exécution.

L'objectif de la planification est de minimiser la fonction de coût suivante : $28 \cdot t_1 + 14 \cdot t_2 + 4 \cdot t_3 + f$, où t_k représente la date de fin la plus tardive parmi celles des interventions de priorité k et f représente la date de fin de l'ensemble des interventions. La date de début d_i (resp. date de fin f_i) d'une intervention I_i s'obtient comme $j_i \cdot H + h_i$ (resp. $j_i \cdot H + h_i + D(i)$), les jours étant numérotés à partir de 0.

La définition du problème peut encore être étendue de deux manières. La première est l'introduction de relations de précédence entre les interventions ; pour toute intervention I_i, une liste $P(i)$ d'interventions devant précéder l'exécution de I_i peut être précisée (c'est-à-dire que toute intervention $I_{i'} \in P(i)$ doit satisfaire $f_{i'} \leq d_i$). Notons que les éventuels temps de latence entre interventions (déplacements, pauses, *etc*) sont ici considérés comme nuls. La seconde extension est liée à l'existence d'un budget B permettant de sous-traiter un certain nombre d'interventions. Ainsi, un coût de sous-traitance $S(i)$ est défini pour toute intervention I_i et la somme des coûts $S(i)$ de toutes les interventions sous-traitées (c'est-à-dire non planifiées) doit être inférieure au budget B. Afin de garantir le respect des précédences dans ce cas, toute intervention I_i sous-traitée entraîne récursivement la sous-traitance des interventions $I_{i'}$ telles que $I_i \in P(i')$.

1.2 État-de-l'art et contribution

À notre connaissance, ce problème n'a pas été abordé en l'état dans la littérature, tant du point de fondamental qu'expérimental. Du fait de sa généricité, le problème contient plusieurs sous-problèmes NP-difficiles. Pour une partition des techniciens en équipes un jour donné, déterminer si un ensemble d'interventions peut être affecté à cet ensemble d'équipes tout en respectant la durée H de la journée, les contraintes de précédence et les contraintes de compétences est NP-complet, même si la durée d'exécution des interventions est unitaire (toutes les interventions ont la même durée d'exécution), le nombre d'équipes fixé à deux et le graphe des précédences isomorphe à un ensemble de chemins orientés sommets-disjoints [18]. Dans le cas de précédences quelconques, le problème reste NP-complet, même si la durée d'exécution des interventions est unitaire et les contraintes de compétences omises (une intervention peut être effectuée par n'importe quelle équipe) [10]. Minimiser le nombre de jours nécessaires à l'exécution de toutes les interventions est NP-difficile au sens fort, même si les interventions peuvent être effectuées par une seule et même équipe chaque jour, sans contrainte de précédences ni de compétences. En effet, ce sous-problème se ramène au problème de *bin-packing* [10], lorsque la durée H n'est pas une constante du problème. Enfin, maximiser la somme des durées de l'ensemble des interventions sous-traitées est équivalent à un problème de *knapsack* (avec contraintes de précédence) [10].

Ce problème, de par sa complexité et son échelle (des centaines d'interventions et de techniciens), est typique des grands problèmes industriels d'optimisation combinatoire. Dans cette note, nous décrivons une heuristique à base de recherche locale pour le résoudre de manière effective et efficace. Notre algorithme a été classé 2ème du Challenge ROADEF 2007 toutes catégories confondues (sur les 35 équipes ayant soumis leurs travaux) [3]. L'algorithme victorieux, dû à Hurkens [16], peut être vu comme une heuristique à base de recherche locale où de larges voisinages sont explorés par programmation linéaire en nombres entiers (à l'aide du solveur ILOG CPLEX 10.0) ; Cordeau *et al.* [2], classé 2ème *ex æquo*, ont également développé une approche par recherche locale à voisinages larges, mais basée sur des mouvements de type destruction/réparation. Avant de décrire notre algorithme, nous introduisons dans ses grandes lignes la méthodologie que nous avons suivie pour concevoir et implémenter celui-ci. Cette méthodologie, initiée lors

de notre participation au précédent Challenge ROADEF 2005 [4, 5, 6, 7] que nous avions alors remporté, est susceptible d'intéresser les chercheurs et ingénieurs proches de la communauté "contraintes" (la recherche locale est par exemple au cœur du résolveur de contraintes *Comet* [20]).

Pour une introduction au paradigme de la recherche locale et des exemples d'applications, le lecteur peut consulter le livre édité par Aarts and Lenstra [1].

2 Méthodologie

Nous insistons ici sur la notion de performance, centrale dans notre approche de la recherche locale. C'est pourquoi avant de décrire la méthodologie sur laquelle nous nous appuyons, voici quelques détails sur ce que nous entendons par algorithmique haute performance.

2.1 Algorithmique haute performance

Nous considérons comme performant un algorithme capable de fournir en temps raisonnable (quelques minutes) une solution de qualité. La plupart des besoins en optimisation rencontrés dans l'industrie s'exprime en ces termes ; lorsque les instances considérées sont de très grandes tailles (par exemple un voyageur de commerce parcourant plus d'un million de villes), un temps d'exécution de plusieurs heures peut être envisagé. Un algorithme performant doit également répondre à certaines exigences en matière d'ingénierie logicielle (fiabilité, robustesse, portabilité, maintenabilité).

La notion de haute performance est souvent assimilée à l'implémentation d'algorithmes sur une architecture à haute performance (en particulier sur une architecture parallèle). En fait, un algorithme haute performance reprend simplement la définition que nous avons donné dans le paragraphe précédent, en y ajoutant des superlatifs : algorithme capable de fournir une solution de très grande qualité en un temps très court. De même, l'exigence de robustesse est redoublée : un tel algorithme doit être capable de traiter convenablement des instances pathologiques. Pour plus de détails sur la notion de haute performance, nous invitons le lecteur à consulter les articles de Moret [21, 22] et les travaux remarquables d'Helsgaun [13, 14, 15] sur le problème du voyageur de commerce.

2.2 Une conception à 3 niveaux

Nous préconisons d'aborder la conception d'une heuristique à base de recherche locale suivant un cadre à trois niveaux : 1) stratégie de recherche, 2) mouvements, 3) algorithmique & implémentation. Nous estimons que la performance d'une heuristique à base de recherche locale est fonction *à part égale* du bon traitement de chacun de ces trois niveaux. En fait, ces trois niveaux couvrent les deux aspects fondamentaux de la recherche locale : la définition de l'espace des solutions (densité + connexité) et l'exploration de cet espace.

La stratégie de recherche, spécifique au problème traité, permet de redéfinir si nécessaire l'espace des solutions et la fonction objectif. En particulier, relâcher certaines contraintes du problème en les basculant dans la fonction objectif augmente la densité (et *a fortiori* la connexité) de l'espace des solutions, ce qui favorise la convergence de la recherche locale (les solutions non admissibles peuvent être vues comme des points de l'espace sur lesquels s'appuie la recherche afin d'atteindre des solutions admissibles de meilleur coût). Il semble préférable de relâcher en priorité les contraintes les moins dures du problème ; relâcher des contraintes qui structure fortement la solution du problème permet une grande diversification de la recherche, mais rend d'autant plus difficile la convergence vers une nouvelle solution admissible. C'est également à ce niveau que peuvent être intégrées des fragments de métaheuristiques (seuils de dégradation, listes tabou, *etc*). Toutefois, nous pensons que la diversification des solutions au cours de la recherche doit être le fait de la (re)définition de l'espace de recherche (densité) et de la définition des mouvements (connexité), et non pas seulement grâce à une "méta-stratégie", chose qui est rarement mentionnée dans la littérature. C'est pourquoi nous préférons, au moins dans un premier temps, implémenter une simple stratégie du type *first-improvement descent* [1] avec choix stochastique des mouvements. Notons que l'introduction d'éléments stochastiques dans toute procédure de choix apparaît comme favorisant la diversification, et donc la convergence vers un optimum local de meilleure qualité (stochastique ne signifiant pas pour autant uniforme).

Comme nous l'avons mentionné précédemment, les mouvements sont au cœur de l'approche par recherche locale. Clairement, la connexité de l'espace des solutions, déterminante pour la convergence, est fonction de la diversité des mouvements employés. Dans un contexte de haute performance, des mouvements basiques de type échange ou insertion ne suffisent plus : un grand nombre de mouvements, plus ou moins orthogonaux, plus ou moins larges, plus ou moins spécialisés, doivent être définis. En particulier, en s'appuyant sur des propriétés structurelles spécifiques au problème (et parfois même aux instances), il est possible d'augmenter la probabilité de succès d'un mouvement (voir par exemple le travail d'Helsgaun [13, 14, 15] sur le voyageur de commerce ou [6, 7] pour le problème d'ordonnancement de véhicules). Notons que cette idée

naturelle consistant à employer systématiquement un pool de mouvements au cours de la recherche est présente dans les métaheuristiques *Iterated Local Search* et *Variable Neighborhood Search* [12].

Enfin, l'algorithmique, en particulier liée à l'évaluation des mouvements, est déterminante quant à l'effectivité de l'algorithme. La recherche locale étant une technique de recherche incomplète, son effectivité dépend fortement du nombre de solutions visitées dans le temps imparti. Bien que triviale, gardons à l'esprit l'analogie suivante : un pilote quelconque peut gagner en concourant avec un moteur de formule 1, mais le meilleur des pilotes ne gagnera jamais en concourant avec un moteur quelconque. Ainsi, une algorithmique incrémentale, basée sur l'exploitation d'invariants dans les structures de données, peut diviser le temps de convergence par 10 (voir par exemple les travaux de Van Hentenryck [20] réalisés autour du logiciel *Comet*). De même, une implémentation travaillée, soucieuse du principe de localité présidant la gestion de la mémoire cache et optimisée par *profiling* du code, peut encore accélérer par 10 la convergence (voir par exemple les travaux réalisés autour des solveurs SAT [24]). En définitive, il n'est pas surprenant de constater un facteur 100, voire 1000, entre les temps de convergence de deux heuristiques à base de recherche locale relevant pourtant des mêmes principes.

3 Description de l'algorithme

3.1 L'heuristique générale

L'heuristique générale se décompose en quatre phases successives, chaque phase k s'intéressant à la planification des interventions de priorité k. L'objectif d'une phase k est de minimiser la date de fin t_k des interventions de priorité k, sans dégrader les dates de fin des interventions de priorité $k' < k$. Pour cela, un algorithme glouton complète la solution admissible héritée de la phase précédente à l'aide des interventions de priorité k. Ensuite, cette solution est modifiée par recherche locale de manière à diminuer t_k tout en maintenant les dates de fin $t_{k'}$ pour chaque priorité $k' < k$. La procédure de recherche locale, qui a pour finalité de "tasser" un ensemble d'interventions d'une priorité donnée, est centrale dans notre approche.

Plus précisément, l'étape de minimisation d'une date de fin t_k se fait comme suit. Étant donnée une solution admissible avec comme dates de fin $t_1, t_2, \ldots, t_k$, une nouvelle solution admissible avec comme dates de fin $t_1, t_2, \ldots, t_k - 1$ est recherchée. Au cours de la recherche, une intervention est dite non satisfaite si elle dépasse la date de fin $t_k - 1$ ou si l'équipe de techniciens à laquelle celle-ci est affectée ne possède pas toutes les compétences pour la réaliser (nous avons ici un exemple précis de densification de l'espace de recherche : le relâchement de la contrainte de compétences qui porte sur les équipes de techniciens). Dans ce cadre, l'objectif de la recherche locale est de minimiser le nombre d'interventions non satisfaites. Lorsqu'elle parvient à toutes les satisfaire, une nouvelle solution admissible est obtenue et le processus est réitéré. Il est à noter que la contrainte de compétence des équipes est relâchée afin de densifier l'espace des solutions sur lequel travaille la recherche locale, et donc de permettre une plus grande diversification.

Une phase de prétraitement a été adjointe à l'heuristique générale de manière à pouvoir gérer les phénomènes de compensation entre les différents termes de la fonction de coût. En effet, dans certains cas, placer les interventions d'une priorité k avant les interventions d'une priorité $k' < k$ peut être plus favorable en terme de coût, les coefficients de la fonction objectif étant peu discriminants. Cette incohérence dans le modèle mathématique introduit une difficulté supplémentaire, qui en définitive est la plus grande dans le cadre de notre approche par recherche locale : déterminer l'ordre de planification des priorités. Ainsi, la phase de prétraitement cherche à "deviner" cette ordre. Pour cela, nous ordonnançons séparément les interventions de chaque priorité k de façon à déterminer une borne supérieure de leur durée de complétion c_k ; concrètement, ceci est fait par la procédure de recherche locale dans un délai très court (quelques dizaines de secondes). L'ordre effectif de planification des priorités sera celui qui minimise la fonction objectif décrit précédemment, les dates de fin t_k s'obtenant par sommation des durées c_k dans l'ordre considéré. Par exemple, admettons que l'on ait $c_1 = 1200$, $c_2 = 120$, $c_3 = 600$. L'ordre naturel de planification des priorités induit les dates de fin $t_1 = 1200$, $t_2 = 1320$, $t_3 = 1920$, soit un coût de 59760. Or, en inversant la planification des priorités 1 et 2, nous obtenons $t_1 = 1320$, $t_2 = 120$, $t_3 = 1920$, soit un coût de 46320. Par conséquent, l'ordre de planification des priorités retenu pour l'application de l'heuristique générale sera : 2, 1, 3, 4.

Les expérimentations numériques, présentées en fin de papier, montrent que l'ordre optimal de planification des priorités est différent de l'ordre naturel $1, 2, 3, 4$ pour plus de la moitié des instances. L'ordre qui apparaît le plus souvent après $1, 2, 3, 4$ est $2, 1, 3, 4$, ce qui est logique compte tenu des coefficients de la fonction objectif. Par conséquent, une phase de prétraitement pragmatique aurait pu être de lancer l'heuristique générale sur chacun des deux ordres $1, 2, 3, 4$ et $2, 1, 3, 4$ sur une courte période de temps, afin d'identifier le plus prometteur. Nous ne nous attarderons pas d'avantage sur cet aspect du problème, puisque

notre travail s'est essentiellement porté sur la procédure de recherche locale. Mettre en œuvre une heuristique à base de recherche locale travaillant de front sur la fonction objectif globale du problème (et donc l'ensemble des interventions) est envisageable, mais beaucoup plus complexe. Il est à noter que le traitement de fonctions de coût à objectifs multiples (notamment sans ordre lexicographique) reste une difficulté majeure dans la conception d'algorithmes de recherche locale.

3.2 Les mouvements

La recherche locale, employée pour tasser les interventions durant chaque phase, consiste à exécuter de façon stochastique un ensemble de mouvements modifiant la solution courante. Ces mouvements sont de deux types, déplacement ou échange, et de deux catégories, les uns agissent sur les techniciens, les autres sur les interventions. Un mouvement est accepté à condition que la solution obtenue respecte les contraintes de précédence entre interventions, la durée maximale H d'une journée de travail et que le nombre d'interventions non satisfaites ne soit pas augmenté. Comme nous l'avons évoqué plus haut, l'efficacité de la recherche locale repose essentiellement sur les deux points suivants : limiter le taux de rejet de chacun des mouvements et accélérer le processus d'évaluation de chaque mouvement. Ce second point est abordé dans la section suivante, où sont détaillés les aspects algorithmiques de la recherche locale.

Nous avons défini 8 mouvements de base, qui forme le véritable moteur de la recherche locale :

- `Déplacer_Technicien` ;
- `Échanger_Techniciens` ;
- `Déplacer_Intervention_Inter_Journées` ;
- `Déplacer_Intervention_Intra_Journée` ;
- `Déplacer_Intervention_Intra_Équipe` ;
- `Échanger_Interventions_Inter_Journées` ;
- `Échanger_Interventions_Intra_Journée` ;
- `Échanger_Interventions_Intra_Équipe`.

Les mouvements sur les techniciens consistent à déplacer ou échanger des techniciens au sein d'une journée du planning. Les mouvements sur les interventions consistent à déplacer ou échanger des interventions du planning ; les suffixes `Inter_Journées`, `Intra_Journée`, `Intra_Équipe` signifient respectivement que les interventions sont déplacées ou échangées au sein de journées différentes, au sein d'une même journée, au sein d'une même équipe. Ces 8 mouvements sont déclinés sous des formes spécifiques afin d'augmenter leur probabilité de succès (on peut voir cela comme un raffinement des voisinages au sein desquels est effectuée la recherche). Ainsi, pour chaque mouvement, nous avons défini les 3 formes suivantes :

- `Générique` : techniciens ou interventions sont choisis aléatoirement ;
- `Journée_Non_Satisfaite` : techniciens ou interventions sont choisis aléatoirement dans une journée où apparaît au moins une intervention non satisfaite ;
- `Équipe_Non_Satisfaite` : techniciens ou interventions sont choisis aléatoirement dans une équipe où apparaît au moins une intervention non satisfaite.

Enfin, nous avons introduit des mouvements dédiés aux deux possibles extensions du problème que sont l'introduction de précédences entre les interventions et la présence d'un budget permettant d'abandonner des interventions à des sous-traitants :

- `Abandonner_Intervention_Budget` : abandonne une intervention (décliné en `Générique`, `Journée_Non_Satisfaite`) ;
- `Échanger_Intervention_Budget` : échange une intervention abandonnée avec une intervention du planning (décliné en `Générique`, `Journée_Non_Satisfaite`) ;
- `Réintégrer_Intervention_Budget` : réintègre une intervention dans le planning ;
- `Échanger_Intervention_Précédences` : échange deux interventions $I_i, I_{i'}$ telles que $f_i \leq d_{i'}$ et le nombre de descendants de $I_{i'}$ dans le graphe des précédences est supérieur ou égal au nombre de descendants de I_i (décliné en `Inter_Journées`, `Intra_Journée`).

En définitive, nous utilisons un pool de 31 mouvements. La vitesse de convergence de la recherche locale dépend fortement du taux d'emploi de chacun de ces mouvements. Ce taux d'utilisation suit une distribution que nous avons fixée après une expérimentation minutieuse sur les instances des bases A et B fournies par France Télécom. Voici les grandes lignes de cette distribution : 25 % de mouvements `Déplacer_Intervention_Inter_Journées` en `Journée_Non_Satisfaite`, 25 % de mouvements `Déplacer_Technicien` en `Équipe_Non_Satisfaite`, 12 % de mouvements `Échanger_Technicien` en `Équipe_Non_Satisfaite`, 0.2 % à 3 % pour les 28 autres mouvements. Le taux de succès de l'ensemble des mouvements (c'est-à-dire le nombre de mouvements acceptés divisé par le nombre de mouvements tentés) varie entre 10 % et 50 % au cours de la recherche, selon les instances considérées.

3.3 Algorithmique & implémentation

Le schéma d'exécution d'un mouvement est le suivant : si l'évaluation du mouvement est positive (*evaluate*), alors le mouvement est accepté (*perform*). En effet, le nombre de mouvements tentés étant largement

supérieur au nombre de mouvements finalement acceptés, l'évaluation des mouvements apparaît comme critique dans tout algorithme à base de recherche locale. La procédure d'évaluation est elle-même étagée de manière à couper au plus tôt en cas d'échec du mouvement. Ainsi, les différents tests composant celle-ci sont ordonnées en fonction de leur complexité et de leur propension à échouer. Par exemple, les contraintes de précédence étant dures, tous les tests ayant attrait aux respect des précédences lors de l'évaluation des mouvements `Déplacer_Intervention` et `Échanger_Intervention` sont effectués en premier lieu. Ne pouvant détailler ici toute l'implémentation mise en œuvre pour accélérer le processus d'évaluation pour chacun des 8 mouvements de base, nous insisterons sur deux points qui nous paraissent cruciaux : l'évaluation des compétences et la gestion des précédences.

3.3.1 L'évaluation des compétences

Tout mouvement qui touche aux techniciens ou aux interventions d'une équipe nécessite une évaluation de l'adéquation entre les compétences fournies par les techniciens et les compétences requises par les interventions de l'équipe. Pour réaliser cette évaluation, nous associons à chaque équipe de techniciens une matrice C_e de compétences qui donne pour chaque domaine d et niveau l, le nombre de techniciens de niveau au moins l dans le domaine d. De façon symétrique, une matrice C_i est associée à chaque intervention décrivant le nombre de techniciens de niveau au moins l requis dans chaque domaine d. Ainsi, une intervention I_i affectée à l'équipe E_e est non satisfaite (du point de vue des compétences) s'il existe un couple (d, l) tel que $C_i(d, l) > C_e(d, l)$. Le nombre de domaines et de niveaux de compétence n'étant pas borné (par exemple, l'instance B4 des jeux de données fournis par France Télécom comporte 40 domaines de compétences), il est difficile de concevoir une structure de données plus efficace que cette matrice domaines/niveaux pour évaluer les compétences. De fait, l'évaluation de l'impact d'un mouvement sur les compétences s'avère coûteuse en temps dans le pire des cas, puisque en $O(d \cdot l)$.

En veillant à réduire le parcours de cette matrice aux seules cases pour lesquelles un test est nécessaire, il est possible de réduire considérablement le coût de cette évaluation en pratique. Par exemple, lors du déplacement d'une intervention, il est possible de ne parcourir que les domaines utiles de la matrice des compétences requises par cette intervention, c'est-à-dire les domaines pour lesquels au moins un technicien est requis quelque soit le niveau considéré. Ensuite, pour chaque domaine d utile, on peut se restreindre à un intervalle de niveaux. Rappelons que nos matrices de compétences sont construites de manière cumulative : pour chaque domaine, le nombre de techniciens requis par niveau croissant est décroissant. Ainsi, l'évaluation peut débuter au plus haut niveau l_{inf} tel que $C_i(d, l_{\text{inf}}) = C_i(d, l)$ pour tout $l \leq l_{\text{inf}}$ et se terminer au plus bas niveau l_{sup} tel que $C_i(d, l_{\text{sup}}) = 0$. Enfin, un test heuristique moins coûteux en temps peut être effectué en amont du parcours de la matrice, de manière à couper au plus tôt en cas d'évaluation négative. Pour chaque domaine d, définissons $C_e(d) = \sum_l C_e(d, l)$ et de façon symétrique $C_i(d) = \sum_l C_i(d, l)$. Nous avons alors la condition nécessaire suivante : s'il existe un domaine d tel que $C_i(d) > C_e(d)$, alors I_i est non satisfaite (notons bien que la réciproque est fausse). Ce test placé en amont permet, en cas d'échec, de déterminer en temps $O(d)$ seulement le statut de l'intervention. De la même manière, il est opportun de placer encore en amont un test vérifiant si $C_i = \sum_d C_i(d)$ est strictement supérieur à $C_e = \sum_d C_e(d)$. En définitive, l'évaluation des compétences comprend une succession de 3 tests, respectivement en temps $O(1)$, $O(d)$ et $O(d \cdot l)$, chacun permettant de conclure en cas d'échec. Bien entendu, toutes ces structures employées pour l'évaluation doivent être maintenues de façon incrémentale au cours de la recherche.

3.3.2 La gestion des précédences

Le second point concerne l'évaluation du dépassement de la durée H de chaque journée de travail ainsi que des dates de fin $t_1, t_2, \ldots, t_k - 1$. En effet, cette opération est compliquée par la présence de précédences entre les interventions, car nécessitant un calcul des plus longs chemins (PLC) dans un graphe acyclique orienté (DAG, de l'anglais *directed acyclic graph*). Pour cela, nous attachons un DAG à chaque journée du planning. Chaque DAG contient un nœud source marquant le début de la journée et un nœud destination marquant la fin de la journée. À chaque intervention planifié dans la journée est associé un nœud dans le DAG. Ces nœuds sont reliés entre eux par des précédences de deux types : des arcs bleues matérialisant l'ordre dans lequel sont planifiées les interventions de chaque équipe au sein de la journée et des arcs rouges représentant les contraintes données en entrée du problème. La longueur $l(i, i')$ de l'arc reliant les nœuds correspondant à deux interventions $I_i \prec I_{i'}$ est donné par la durée $D(i)$ de l'intervention I_i. De cette façon, la date de début au plus tôt d'une intervention est donnée par la longueur du PLC allant à son nœud dans le DAG. Cette date, stockée à chaque nœud, permet de vérifier le respect de la durée maximale H des journées de travail et des dates de fin $t_1, t_2, \ldots, t_k - 1$, et donc d'évaluer le nombre d'interventions non satisfaites.

Ainsi, tout mouvement `Déplacer_Intervention`

ou `Échanger_Intervention` entraînent un cascade d'ajouts/suppressions d'arêtes dans les DAG des journées concernées, nécessitant une mise à jour (temporaire) des PLC afin d'évaluer l'impact du mouvement. Les interventions de chaque équipe étant réalisées de manière séquentielle, chaque nœud possède un unique prédécesseur bleu et un unique successeur bleu ; les prédécesseurs et les successeurs rouges sont eux stockés dans des tableaux au niveau du nœud. Le choix de cette représentation est motivé par la faible densité du graphe des précédences constatée sur les instances A et B (où le nombre d'arcs rouges est inférieur au nombre n d'interventions). Cette mise à jour temporaire des PLC est réalisée par une propagation récursive en largeur à partir du nœud inséré ou supprimé : le nouveau PLC au nœud en question est calculé en parcourant ses prédécesseurs, si ce nouveau PLC est différent de l'ancien, alors les successeurs du nœud sont placés dans une file afin d'être examinés récursivement. Cette propagation permet également de détecter la création de cycles, invalidant le mouvement. Dès lors que les degrés maximums entrant et sortant d'un nœud sont en $O(1)$, ce qui s'avère être le cas ici, notre algorithme incrémental s'exécute en temps optimal $O(a)$, avec a le nombre de nœuds affectés (c'est-à-dire ceux dont le PLC est modifié). Nous invitons le lecteur intéressé à consulter les travaux de Katriel *et al.* [19] sur le sujet, qui proposent un algorithme incrémental dont la complexité se révèle avantageuse lorsque le degré maximum entrant d'un nœud est grand.

3.3.3 Un détail d'implémentation

Comme nous l'avons évoqué en introduction, introduire des éléments stochastiques dans les procédures de choix apparaît comme déterminant dans le processus de diversification. Ainsi, outre le choix du mouvement lui-même, tous les choix nécessaires à l'exécution d'un mouvement sont aussi stochastiques. En moyenne, la fonction `MyRand(n)`, qui retourne un entier (pseudo) aléatoire entre 0 et $n-1$, est appelée 5 fois par mouvement. Par exemple, le mouvement `Déplacer_Intervention_Inter_Journées` décliné en `Journée_Non_Satisfaite` (qui représente 25 % des mouvements tentés) comporte 6 appels à la fonction `MyRand`. En définitive, cette dernière est de loin la plus appelée du programme (plus de 10 milliards d'appels sur 20 mn d'exécution).

Une implémentation naïve (en langage C ISO) de `MyRand` est `rand() % n`, ou dans une version plus robuste `n * rand() / (RAND_MAX + 1.0)`, où `rand()` est une fonction de la bibliothèque `stdlib` retournant un nombre pseudo-aléatoire entre 0 et le plus grand entier positif de type `int` (pour tout détail à propos du langage C et ses bibliothèques, nous renvoyons le lecteur à [11]). Bien que produisant des séquences de nombres pseudo-aléatoires de qualité suffisante pour notre application, une telle implémentation est relativement lente (due à la présence d'une opération de division). Plus précisément, un *profiling* de notre programme à l'aide du logiciel *gprof* [9] identifiait `MyRand` comme un goulot d'étranglement majeur au niveau du temps d'exécution. Inspirés du générateur de Knuth et Lewis [23, pp. 274–286], nous avons écrit notre propre fonction `MyRand(n)` optimisée : `(n * ((seed = 1664525 * seed + 1013904223)` $\gg$ `16))` $\gg$ `16`, qui fonctionne à condition que n soit compris entre 0 et $2^{16}-1 = 65535$ et que le type `int` soit codé sur 32 bits (`seed` est la graine du générateur, initialisée en début du programme).

En moyenne, cette implémentation, qui comprend seulement 2 multiplications, 2 décalages, 1 addition et 1 affectation, est 3.5 fois plus rapide que l'implémentation naïve (la qualité des séquences de nombres pseudo-aléatoires retournées étant comparable). Elle nous a permis de réduire la part du temps total d'exécution consommé par la fonction `MyRand` de 17 % à 7 %, la ramenant ainsi au même niveau que d'autres fonctions clés du programme (les 3 fonctions les plus coûteuses après `MyRand` consomment chacune autour de 5 % du temps total d'exécution).

4 Résultats expérimentaux

L'algorithme a été implémenté en langage C ISO (C99). Le programme résultant, qui compte environ 12000 lignes de code, a été compilé et testé sous différentes architectures aux performances comparables (entre autres : Red Hat Linux sur AMD Athlon 64, Windows XP sur Intel Pentium 4, Windows Vista sur Intel Xeon) à l'aide du compilateur `GCC 3.4.4` (avec les options `-O3 -pedantic -Wall -W -std=c99 -march={athlon64,pentium4,nocona}`). Les jeux de données A, B et X utilisés pour les tests, fournis par France Télécom, peuvent être téléchargés sur la page internet du Challenge [3] (le jeu X, qui a servi à l'évaluation des algorithmes soumis par les participants, n'a été dévoilé qu'une fois le Challenge terminé). Quelque soit l'architecture, notre implémentation permet la tentative de plus d'un million de mouvements par seconde, et ce même dans le cas d'instances volumineuses (par exemple l'instance B8 : 800 interventions, 150 techniciens, 10 domaines de compétence, 4 niveaux, 440 précédences) ; sur 20 minutes d'exécution, l'algorithme dépasse le milliard de solutions visitées. La mémoire allouée par notre programme n'excède pas 10 Mo quelque soit l'instance considérée (par exemple, 8 Mo sont allouées pour l'instance B8), permettant

une pleine exploitation de la mémoire cache. Les tableaux ci-dessous (Fig. 1, 2, 3, 4, 5, 6) regroupent les résultats obtenus sur un ordinateur équipé d'un système Windows XP et d'un processeur Intel Xeon 3075 (2 cœurs CPU 2.67 GHz, cache L1 2×64 Ko, cache L2 4 Mo, RAM 2 Go). Un code exécutable du programme (compilé pour l'architecture cible souhaitée) peut être obtenu auprès des auteurs sur demande.

data	n	m	d	l	P	B
A1	5	5	3	2	0	0
A2	5	5	3	2	2	0
A3	20	7	3	2	0	0
A4	20	7	4	3	7	0
A5	50	10	3	2	13	0
A6	50	10	5	4	11	0
A7	100	20	5	4	31	0
A8	100	20	5	4	21	0
A9	100	20	5	4	22	0
A10	100	15	5	4	31	0

FIG. 1 – Instances A : caractéristiques.

data	FT	EGN	BEST	% écart	ordre
A1	2490	2340	2340	0.0	1234
A2	4755	4755	4755	0.0	1234
A3	15840	11880	11880	0.0	2134
A4	14880	14040	13452	4.4	1234
A5	41220	29400	28845	2.0	2134
A6	30090	18795	18795	0.0	2134
A7	38580	30540	29690	2.9	1234
A8	26820	20100	16920	18.8	1234
A9	35600	27440	27440	0.0	2134
A10	51720	38460	38296	0.5	1234

FIG. 2 – Instances A : résultats.

Chaque jeu d'instances est présenté à l'aide de deux tableaux. Dans le premier tableau, nous donnons pour chaque instance la taille des données clés en entrée : le nombre n d'interventions, le nombre m de techniciens, le nombre d de domaines de compétence, le nombre l de niveaux de compétence, le nombre P de précédences (non transitives) entre interventions, le budget B alloué. Pour chaque instance, nous avons avons effectué 5 essais, chacun avec temps d'exécution limité à 1200 secondes (20 mn). Dans le second tableau, les colonnes "FT", "EGN", "BEST", "% écart", "ordre" contiennent respectivement le résultat obtenu par France Télécom, le moins bon résultat obtenu par notre algorithme, le meilleur résultat obtenu parmi les participants au Challenge (et les 5 essais réalisés avec notre algorithme), l'écart relatif (en %) entre les valeurs des deux colonnes précédentes, et l'ordre de planification des priorités utilisé par l'algorithme EGN (par exemple, la valeur 3214 signifie que l'ordre de planification des priorités est 3, 2, 1, 4).

data	n	m	d	l	P	B
B1	200	20	4	4	47	300
B2	300	30	5	3	143	300
B3	400	40	4	4	57	500
B4	400	30	40	3	112	300
B5	500	50	7	4	427	900
B6	500	30	8	3	457	300
B7	500	100	10	5	387	500
B8	800	150	10	4	440	500
B9	120	60	5	5	55	100
B10	120	40	5	5	55	500

FIG. 3 – Instances B : caractéristiques.

data	FT	EGN	BEST	% écart	ordre
B1	69960	33900	33675	0.7	1234
B2	34065	16260	15510	4.9	1234
B3	34095	16005	15870	0.9	1234
B4	50340	24330	23700	2.7	2134
B5	150360	88680	87300	1.6	1234
B6	47595	27675	27210	1.8	2134
B7	56940	36900	33060	11.7	1234
B8	51720	36840	32160	14.6	1234
B9	44640	32700	28080	16.5	1234
B10	61560	41280	34440	19.9	1234

FIG. 4 – Instances B : résultats.

De manière générale, nous constatons un faible écart entre les 5 essais que nous avons réalisés. Notons que les écarts sont d'autant plus grands que le nombre de jours planifiés augmente. Ainsi, des écarts entre essais supérieurs à 1 % sont constatés pour les instances suivantes : X1 (57 jours), X5 (52 jours), X9 (50 jours), X10 (49 jours). L'écart relatif entre les résultats de notre algorithme et les meilleurs résultats du Challenge montre que notre algorithme est très compétitif. En moyenne, l'algorithme EGN obtient un gain de 30.6 % sur le coût des solutions obtenues par France Télécom (41.5 % sur les instances B). D'autre part, nos solutions se situent en moyenne à 6.4 % de la meilleure solution obtenue sur l'ensemble des compétiteurs (avec un écart à la moyenne de 5.8 %). Sur les 30 instances, notre algorithme obtient la meilleure solution en terme de coût pour 13 d'entre elles (7 sur le jeu A, 6 sur le jeu B) et obtient une solution d'un coût inférieur à 6 %

data	n	m	d	l	P	B
X1	600	60	15	4	195	50
X2	800	100	6	6	536	500
X3	300	50	20	3	224	1000
X4	800	70	15	7	321	150
X5	600	60	15	4	201	50
X6	200	20	6	6	128	500
X7	300	50	20	3	235	1000
X8	100	30	15	7	40	150
X9	500	50	15	4	184	50
X10	500	40	15	4	184	500

FIG. 5 – Instances X : caractéristiques.

data	FT	EGN	BEST	% écart	ordre
X1	n/a	180240	151140	19.3	1234
X2	n/a	8370	7260	15.3	1234
X3	n/a	50760	50040	1.5	1234
X4	n/a	68960	65400	5.5	2134
X5	n/a	178560	147000	21.5	1234
X6	n/a	10440	9480	10.2	1234
X7	n/a	38400	33240	15.6	1234
X8	n/a	23800	23640	0.7	1234
X9	n/a	154920	134760	15.0	1234
X10	n/a	152280	137040	11.2	1234

FIG. 6 – Instances X : résultats.

de celui de la meilleure solution pour 18 d'entre elles (9 sur le jeu A, 6 sur le jeu B, 3 sur le jeu X). En outre, nous avons pu identifier les raisons de l'échec de l'algorithme EGN sur les 17 instances pour lesquelles nous n'obtenons pas la meilleure solution. La raison première est le fait que l'ordre retourné de planification des priorités calculé par notre phase de prétraitement n'est pas optimal. En effet, le tableau ci-contre (Fig. 7) montre le coût obtenu par l'algorithme en supposant l'ordre optimal connu. Ce coût est donné dans la colonne marqué "EGN*" et l'ordre optimal est donné dans la colonne "ordre". Dans ce cas, nous constatons que pour 6 instances de plus, nous obtenons la meilleure solution en terme de coût. La seconde raison est également due au caractère multi-objectif de la fonction de coût. Par exemple, pour l'instance A4, l'algorithme EGN obtient la solution suivante : $t_1 = 315$, $t_2 = 540$ et $t_3 = 660$ pour un coût global de 14040. Or, relâcher légèrement la date de fin des interventions de priorité 1 permet d'améliorer le coût global par compensation des deux premiers termes de la fonction objectif : $t_1 = 324$, $t_2 = 480$ et $t_3 = 660$ pour un coût global de 13452 (meilleure solution connue).

En revanche, notre approche par recherche locale

data	EGN	EGN*	BEST	% écart	ordre
A5	29700	28845	28845	0.0	3214
A8	20100	16979	16979	0.0	2134
B7	36900	35700	33300	7.3	2134
B9	32700	28080	28080	0.0	2134
B10	41280	34440	34440	0.0	2314
X2	8370	7440	7260	2.5	2134
X6	10440	10140	9480	7.0	2134
X7	38400	32280	32280	0.0	2134
X8	23800	23220	23220	0.0	2134

FIG. 7 – Résultats obtenus connaissant l'ordre optimal des priorités.

semble dépassée sur les instances X1, X5, X9, X10 par les approches par recherche locale à voisinages larges de Hurkens [16], vainqueur du Challenge, et dans une moindre mesure, de Cordeau *et al.* [2], classé 2ème *ex æquo*. Pour palier à cette faiblesse, il nous faudrait donc ajouter à notre pool de transformations un mouvement à voisinage large (comme nous l'avions fait dans [7] pour le problème d'ordonnancement de véhicules). Ci-dessous (Fig. 8) sont présentés les résultats que nous obtenons pour ces 4 instances en allongeant les temps d'exécution, montrant que notre algorithme converge vers une solution d'une qualité proche de celles qu'obtiennent Hurkens et Cordeau *et al.* [3].

data	20 mn	1 heure	3 heures	9 heures
X1	180240	170460	168240	158280
X5	178560	167280	165120	164760
X9	154920	146520	146040	141720
X10	152280	144360	140340	140160

FIG. 8 – Résultats obtenus avec des temps d'exécution plus longs.

Références

[1] E. Aarts, J.K. Lenstra, eds (1997). *Local Search in Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester, England, UK.

[2] J.-F. Cordeau, G. Laporte, F. Pasin, S. Ropke (2007). ROADEF 2007 Challenge : scheduling of technicians and interventions in a telecommunications company. In *ROADEF 2007, le 8ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (G. Finke, ed.). Grenoble, France.

[3] V.-D. Cung, A.-M. Bustos, A. Laugier, P.-F. Dutot, C. Artigues, O. Briant, S. Oussedik (2007). Challenge ROADEF 2007. `http://www.g-scop.fr/ChallengeROADEF2007/`

[4] V-.D. Cung, A. Nguyen, Y. Khacheni, C. Artigues, C.M. Li, B. Penz (2005). Challenge ROADEF 2005.

[5] V-.D. Cung, A. Nguyen (2005). Le problème du Car Sequencing RENAULT et le Challenge ROADEF'2005. In *Actes des JFPC 2005, les 1ères Journées Francophones de Programmation par Contraintes* (C. Solnon, ed.), pp. 3–10. Lens, France.

[6] B. Estellon, F. Gardi, K. Nouioua. Two local search approaches for solving real-life car sequencing problems. (à paraître dans *European Journal of Operational Research*)

[7] B. Estellon, F. Gardi, K. Nouioua (2006). Large neighborhood improvements for solving car sequencing problems. *RAIRO Operations Research* 40(4), pp. 355–379.

[8] B. Estellon, F. Gardi, K. Nouioua (2007). Une heuristique à base de recherche locale pour la planification de tâches avec affectation de ressources. In *ROADEF 2007, le 8ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (G. Finke, ed.). Grenoble, France.

[9] J. Fenlason, R. Stallman (1998). *GNU gprof : the GNU profiler.*

[10] M.R. Garey, D.S. Johnson (1979). *Computer and Intractability : A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co, San Francisco, CA.

[11] H. Garreta (1992). *C : Langage, Bibliothèque, Applications.* InterÉditions, Paris, France.

[12] F.W. Glover, G.A. Kochenberger (2002). *Handbook of Metaheuristics.* International Series in Operations Research and Management Science, Vol. 57, Kluwer Academic Publishers, Norwell, MA.

[13] K. Helsgaun (1998). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Datalogiske Skrifter (Writings on Computer Science)*, Technical Report No. 81. Roskilde University, Danemark.

[14] K. Helsgaun (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research* 126(1), pp. 106–130.

[15] K. Helsgaun (2006) An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic. *Datalogiske Skrifter (Writings on Computer Science)*, Technical Report No. 109. Roskilde University, Danemark.

[16] C.A.J. Hurkens (2007). Incorporating the strength of MIP modeling in schedule construction. In *ROADEF 2007, le 8ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (G. Finke, ed.). Grenoble, France.

[17] C.A.J. Hurkens (2008). Communication personnelle.

[18] K. Jansen, G.J. Woeginger, Z. Yu (1995). UET-scheduling with chain-type precedence constraints. *Computers and Operations Research* 22(9), pp. 915–920.

[19] I. Katriel, L. Michel, P. Van Hentenryck (2005). Maintaining longest paths incrementally. *Constraints* 10(2), pp. 159–183.

[20] L. Michel, P. Van Hentenryck (2002). A constraint-based architecture for local search. In *Proceedings of OOPSLA 2002, the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, *SIGPLAN Notices* 37(11), pp. 83–100. ACM Press, New York, NY.

[21] B.M.E. Moret (2002). Towards a discipline of experimental algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology : 5th and 6th DIMACS Implementation Challenges* (M.H. Goldwasser, D.S. Johnson, C.C. McGeoch, eds.), DIMACS Monographs, Vol. 59, pp. 197–213. American Mathematical Society, Providence, RI.

[22] B.M.E. Moret, D.A. Bader, T. Warnow (2002). High-performance algorithm engineering for computational phylogenetics. *Journal of Supercomputing* 22(1), pp. 99–111.

[23] W.H. Press, S.A. Tenkolsky, W.T. Vetterling, B.P. Flannery (1995). *Numerical Recipes in C : the Art of Scientific Computing.* Cambridge University Press, Cambridge, Royaume-Uni. (2ème édition)

[24] L. Zhang, S. Malik (2003). Cache performance of SAT solvers : a case study for efficient implementation of algorithms. In *Proceedings of SAT03, the Sixth International Conference on Theory and Applications of Satisfiability Testing* (S.M. Ligure, ed.), pp. 287-298.