Recherche locale en optimisation combinatoire : fondements et applications

Frédéric Gardi, Bouygues e-lab, Paris

Plan :

A) Fondements de la recherche locale

1) Optimisation combinatoire
   a) Problème d'optimisation
   b) Problème de décision
   c) Panorama et histoire des techniques de résolution

2) Recherche locale = recherche dans des voisinages
   a) Fonction de voisinage
   b) Optimum local vs. optimum global
   c) Exploration du voisinage
   d) Questions fondamentales

3) Stratégies de recherche
   a) Stratégies basiques
   b) Stratégies avancées et métaheuristiques

B) Applications industrielles

4) Le problème de l'ordonnancement de véhicules
   a) Définition du problème académique
   b) Voisinages naturels
   c) Recherche locale par petit voisinage
   d) Recherche locale par grand voisinage
   e) Le problème chez RENAULT

5) Planification des interventions et des techniciens chez France Telecom
   a) Définition du problème
   b) Heuristique générale
   c) Recherche locale
   d) Résultats

6) Conclusion : les ingrédients pour une recherche locale efficace
   a) Diversifier les voisinages / la recherche
   b) Accélérer l'exploration / évaluation des voisinages
   c) La clé de la réussite : le « volume » de transformations

Docs : papiers RAIRO, EJOR, ROADEF 2007
Figures : Fig. 9 papier RAIRO, Fig. 3,4,5,6 papier EJOR

B) Applications industrielles

- problème de l'ordonnancement de véhicules (académique + RENAULT)
- problème de la planification des interventions et des techniciens (France Telecom)

Bel exemple récent de la puissance de la LS : l'implémentation améliorée de Helsgaun [1999] de l'heuristique de Lin-Kernighan. Le travail d'Helsgaun est l'exemple d'une implémentation performante d'une LS pour résoudre un problème difficile, le TSP, sur de larges instances.

Le travail d'Helsgaun suit exactement la recette que nous avons donné (pilote + volant + moteur). Il s'est concentré sur deux aspects : i) les voisinages (le volant) et ii) l'algorithmique (le moteur) :

i) mouvements 5-opt spéciaux, nouveaux mouvements non séquentiels, listes de candidats efficaces, utilisation de l'algorithme 1-tree de Held & Karp, …

ii) structures de données très efficaces, calcul des coûts, calcul de borne supérieure sur les coûts des mouvements, analyse sensitive, …

Le résultat est surprenant : son algorithme produit des solutions optimales pour toutes les instances connues de la TSPLIB (y compris les fameuses instances à 7397 et 13509 villes d'Applegate et al.). Temps d'exécution en moyenne estimé à O(n^2.2). Pour l'instance 13509 : quelques minutes sur un PC. Optimalité atteinte par Applegate et al. sur un réseau de + de 60 machines en 3 mois (le + long est la preuve de l'optimalité de la solution). Une meilleure solution a même été trouvée pour l'instance à 85900 villes en 2 semaines de calcul.

Même si le travail d'Helsgaun est de l'ordre d'un travail de recherche, c'est l'exemple même de ce qu'il faut faire pour traiter efficacement un problème pratique : grande diversité dans les voisinages (avec stochastique) + grande efficacité algorithmique/implémentation.

4) Le problème de l'ordonnancement de véhicules

a) Définition du problème académique

Véhicules à ordonner au travers des ateliers de production.
3 ateliers : tôlerie, peinture, assemblage
Automatisation : 80 %, 50 %, 20 %

Problème académique (le + fréquent dans la littérature) s'intéresse au séquencement des véhicules dans l'atelier d'assemblage. L'assemblage du véhicule se fait au travers de plusieurs postes de travail. Chaque poste est dédié à la pose d'une option : air conditionné, toit ouvrant, poste radio, … La chaîne tournant en continu, il est nécessaire de ne pas surcharger un poste en plaçant plusieurs véhicules réclamant la même option à la suite dans la chaîne.

Exemple : le toit ouvrant est un opération lourde (coûteuse en temps) ; si deux véhicules consécutifs à toit ouvrant, la cadence de la chaîne doit être ralentie.

Ce besoin d'espacer les véhicules ayant les mêmes options est formalisé au travers d'une contrainte de ratio pour chaque option. Par exemple, une option avec une contrainte de ratio de 3/7 signifie pas plus de 3 véhicules dans toute fenêtre de taille 7 (fenêtres = ensemble de

positions consécutives). Dans le problème généralement traité dans la littérature, on cherche à trouver une séquence sans violation de contrainte ratio.

Dans la réalité, une telle séquence n'existe pas forcément. On cherche alors à minimiser le nombre de violations sur chaque option. Une violation est comptée pour chaque dépassement du ratio dans chaque fenêtre, même sur les bords.

Exemple : __XXXXX => 2 violations pour 3/7

Etant n véhicules ayant chacun un ensemble d'options parmi k, l'objectif est de trouver une séquence (ou permutation) de ces n véhicules minimisant le nombre de violations pour chaque option.

Notion de classe de véhicules : les véhicules peuvent être groupées par classe, une classe étant déterminée par les mêmes options. Si k options, au maximum sum_{i=1}^{k} C^i_k classes.

Exemple : pour 4 options, 15 classes possibles

C'est un problème NP-difficile [Kis, 2004] : réduction à partir du 3-set cover. Si k est de l'ordre de log(n), algorithme pseudo polynomial basé sur la programmation dynamique mais inefficace en pratique (n=10 ok, n=20 plusieurs minutes).

Beaucoup de travaux pratiques réalisés et publiés autour de ce problème depuis 20 ans (importance industrielle !) mais pas tant que cela sur le plan théorique.

Les approches utilisées :

- exactes : PPC (filtrage et propagation spéciale pour les contraintes de ratio), PLNE récemment (B&B avec règle de branchement spéciale)

- heuristiques : constructive (glouton, fourmis), recherche locale (génétique !)

Renouveau avec sujet du Challenge ROADEF 2005. Problème chez RENAULT plus difficile que le problème académique (multiobjectifs : atelier peinture en +). Beaucoup de travaux autour de la recherche locale (+ recuit, taboo, …) : 55 équipes participantes, de 15 pays. On en parle plus tard.

Bon exemple pour l'application de la recherche locale (fonctions de voisinages nombreux, cf. problèmes d'ordonnancement).

b) Voisinages naturels

Ici toute permutation de véhicules est une solution. Trouver une solution admissible est donc facile. De là, on va voir quels sont les voisinages naturels à explorer.

⇨ faire participer les élèves !

Voisinages utilisés par Gottlieb et Puchta [2002,2003] pour traiter le car sequencing académique ainsi qu'un problème de car sequencing chez un constructeur allemand.

SWAP : le voisinage le + naturel. Echanger 2 véhicules de la séquence (n(n-1)/2 swaps possibles) ;

INSERTION (avant/arrière) : sortir un véhicule et l'insérer plus loin (O(n^2))

REFLECTION : inverser un bloc de la séquence (O(n^2))

Deux variantes intéressantes :

SWAP SIMILAR : swap de deux véhicules ayant des options en commun ; limite le taux d'échec du swap puisque 1 ou 2 options diffèrent seulement + évaluation rapide.

SWAP CONSECUTIF : swap de deux véhicules consécutifs ; faible risque de détérioration (uniquement dans une fenêtre pour chaque option : |__X|X__|) + évaluation rapide).

⇨ Détailler le pourquoi de l'évaluation rapide pour les élèves

Ici deux idées fondamentales apparaissent :

- identifier des transformations (ou restrictions) « intéressantes », c'est-à-dire améliorant certainement le coût, ou plutôt à l'opposé, ne détériorant certainement pas le coût. Idée : limiter le taux d'échec d'un mouvement ;

- identifier des transformations (ou restrictions) « efficaces », c'est-à-dire dont l'apport peut être évalué très rapidement ;

Le mieux : lorsque les deux sont conjugués ensemble.

Bon résultat sur de petites instances (CSPLIB) : meilleur que le glouton, aussi bon que les fourmis (Solnon, 2000), meilleur que la PPC.

Ordre d'idée pour l'implémentation : 100 000 itérations / sec environ (ramené sur un 3GHz).

EGN avons travaillé sur ce problème dans le cadre du Challenge ROADEF 2005 (nous en parlons plus tard), nous avons également publié un papier récemment sur le problème académique. Dans ces papiers, nous parlons de LS à petit voisinage (travail de Gottlieb & Puchta revisité) et de LS à grand voisinage (nouveau !).

c) Recherche locale par petit voisinage

Notre idée : appliquer notre recette (heuristique simple mais voisinages diversifiés + algorithmique puissante) sur le car sequencing académique afin de pouvoir comparer nos résultats avec les nombreux résultats présentés dans la littérature.

Heuristique la plus simple : descente + first-improvement

Construire une séquence initiale (algorithme glouton) ;
Tant que temps limite non dépassé ou nombre de violations > 0 faire
    Choisir une des 4 transformations (et les positions à laquelle on l'applique) ;
    Si la transformation ne détériore pas la séquence courante alors on l'applique ;

Attention : ici ne pas détériorer signifie coût inférieur ou EGAL !

Pourquoi first-improvement ? En fait, si l'on veut formaliser notre recherche locale dans ce contexte, on peut dire qu'à chaque itération, on définit un voisinage de taille 4 * n^2 (toutes les possibilités de réaliser chacune des 4 transformations). On visite ce voisinage (plus ou moins) au hasard et on prend la première solution qui est de meilleur coût ou de coût égal.

- <u>voisinages diversifiés :</u>

Idée : combiner les voisinages de façon stochastique (SVNS : stochastic variable neighborhood search)

On décline les transformations basiques : swap, insertion, reflection.

SWAP : GENERIC : positions tirés au hasard (transformations basiques)
CONSECUTIVE : positions consécutives
SIMILAR : véhicules similaires

FORWARD INSERTION
BACKWARD INSERTION
REFLECTION

GENERIC : idem
DENOMINATOR : positions de façon à ce que le bloc visé ait la taille égale au (multiple du) dénominateur du ratio d'une ou plusieurs options. Limite le taux d'échec : maintien une certaine structure dans les fenêtres associées à ces options. En fait, cela revient à mixer des bouts de fenêtres pour les options visées (début d'analogie avec la génétique !).

Voici les pourcentages utilisés (la meilleure sauce que nous ayons trouvée) :

Swap	generic : 69.6 % consecutive : 3.2 % similar : 2.5 %
Forward insertion	generic : 3.2 % denominator : 3.8 %
Backward insertion	generic : 3.2 % denominator : 3.8 %
Reflection	generic : 6.9 % denominator : 3.8 %

Solutions obtenues de très bonne qualité. Même à coût constant, énormément de solutions générées. Diversification permanente de la recherche et donc des solutions (même lorsque l'on est bas dans la descente). En pratique, convergence vers l'optimum global (pour les instances de la CSPLIB).

Comment trouver la bonne « sauce » entre tous les voisinages ? Pas de recette miracle, à la main, en expérimentant. En particulier, analyser la descente de votre algorithme dans le détail. Quelles sont les transformations qui réussissent ? Qui font progresser la recherche (coût strictement meilleur) ? Cela en mettant une probabilité uniforme sur la sélection de chaque transformation pour commencer.

Une idée peut être aussi de faire cela de façon automatique en faisant évoluer les probabilités de façon dynamique au cours de la recherche. Par exemple, j'augmente les mouvements qui ont du succès, j'augmente plus encore ceux qui font progresser la recherche et je pénalise ceux qui échouent (je compte les succès, les échecs). Bonne idée, mais attention : risque d'instabilité, risque de bannir certaines transformations qui peuvent s'avérer déterminante à certaines moments de la recherche ; solution : (toujours) mettre des seuils pour stabiliser et assurer un minimum de chaque transformation utile.

Ne pas hésiter à dégager les transformations inutiles (taux de réussite quasi nul TOUT LE LONG de la recherche).

Ici, les swaps génériques sont majoritairement utilisés, car puissant pour diversifier la recherche (taux de progression faible, mais taux de réussite important) : assure une grande diversification. Les swaps consécutifs et les reflections ont un taux de réussite et un taux de progression important.

Expérimentations montrent que l'utilisation seule (ou trop élevé) de swaps ne converge pas vers l'optimum global (blocages constatés sur toutes les instances). Alors que converge avec insertions et reflections. Cela signifie qu'il y a certaines solutions que l'on peut atteindre (ou construire) à l'aide de swaps seulement. D'où l'importance de combiner divers voisinages.

Conclusion : diversité de voisinages => diversification assurée => convergence assurée ?

- <u>algorithmique puissante :</u>

Nous venons de voir comment assurer la convergence vers optimum local de qualité (voire même optimum global). Le problème est que si la convergence se fait en plusieurs heures, c'est comme si elle ne se faisait pas ! Le but n'est pas seulement d'obtenir de bonnes solutions, mais surtout les obtenir rapidement ! Par conséquent, la vitesse de convergence est un facteur déterminant.

Attention : ce qui est dommage est d'avoir les bons voisinages, mais de croire qu'il n'y a pas de convergence parce que l'implémentation est mauvaise et donc la convergence lente.

On comprend donc que l'efficacité algorithmique (voire l'optimalité algorithmique) de l'exploration est nécessaire pour évaluer une heuristique et surtout des voisinages.

Ici, les 3 transformations ont de très bonnes propriétés. En effet, chacune ne modifie qu'au plus 2.Q_i fenêtres pour chaque option i.

⇨ montrer cela au élève au tableau

Cela montre bien que ces transformations ne modifie la séquence que très localement, car en effet, on a Q_i << n. Deux conséquences :

- c'est en fait la raison fondamentale pour laquelle ces transformations ont un taux d'échec limité ;
- cela permet une évaluation de l'impact de chacune en O(Q_i) par option grâce à des structures de données spéciales s'appuyant sur les invariants. Ici la structure de base est la fenêtre associée à chaque option, on retient donc le nombre de véhicules ayant l'option dans chaque fenêtre et pour chaque option. De là, l'évaluation des transformations peut se faire efficacement (assez simple pour le swap et l'insertion, plus difficile pour la reflection).

Résultat : non pas 100 000 transformations / sec (Gottlieb & Puchta) mais 1 million / sec, soit 10 fois plus. Donc convergence 10 fois plus rapide (et même plus, facteur multiplicatif). Les résultats sont très nets, là où ils convergent en plusieurs dizaine de secondes, on converge en moins d'une seconde.

On aurait pu imaginer utiliser d'autres transformations tel que le BLOCK SWAP ou BLOCK INSERTION. Nous avons remarqué que ces transformations n'apportaient rien. Explication : ces transformations, qui restent dans le domaine du petit voisinage (O(n^2)), déstructure/perturbe plus la séquence et sont plus difficiles à évaluer (nombre de fenêtres touchées + grand). Donc taux d'échec + grand et évaluation + lente. Par conséquent, apport négatif, SAUF si ces transformations sont spécialisées au travers d'un choix des positions plus fins où elles sont appliquées (ex : DENOMINATOR).

Sur des problèmes plus larges à 200, 300, 400, 500 véhicules avec 5 à 7 options, c'est-à-dire proche de la réalité (RENAULT : de qq. centaines jusqu'à 2000 véhicules / jour), nous obtenons les meilleurs résultats / temps de réponse jamais obtenus.

Ex : CSPLIB

17/39 instances à l'optimum (connu ou supposé) en – d'1 sec.
33/39 en – de 10 sec.

+ Amélioration de 3 bornes

Instance 300-05 : record à 29, nous obtenons 28 en 2mn, et même 27 en 20mn. (Ce sont des moyennes, de grands écarts de temps sont observés pour 27.)

Au final, les autres algorithmes mettent 10 à 100 fois plus de temps pour obtenir les mêmes ou moins bons résultats. En particulier, nous battons les fourmis (françaises et canadiennes), en dépit de techniques très avancées et de couplages avec mouvements LS, ainsi que toutes les approches de type exact.

d) Recherche locale par grand voisinage

Méthode expérimentée lors de la 1ère phase du Challenge. Basée sur une extension du swap (vu comme une 2-permutation) : la k-permutation. Nous ne définissons pas le voisinage comme toutes les k-permutation de véhicules, mais plus simplement comme l'ensemble des solutions atteignables en permutant les véhicules positionnées sur k positions choisies.

k positions choisies => k! permutations donc exponentiel lorsque k de l'ordre de n (en fait déjà grand pour k = 15 même fixé, + de 1000 milliards (10^12))

Comment explorer intelligemment et efficacement un tel voisinage ? Réponse : par des techniques de PLNE (B&B). En fait, le problème du car sequencing académique possède une formulation compacte en PLNE.

Détailler le PLNE (cf. article RAIRO).
Détailler le nombre de variables et de contraintes (cf. article RAIRO).

Ce PLNE est très difficile à résoudre exactement lorsque le nombre de véhicules dépasse 100 pour quelques options.

Par contre, pourquoi ne pas utiliser le PLNE pour explorer le voisinage à base de k-permutation : ILP-based local search.

Construire une séquence initiale (algorithme glouton) ;
Tant que temps limite non dépassé ou nombre de violations > 0 faire
    Choisir K voitures mobiles ;
    Construire PLNE restreint à ces K véhicules mobiles (fixation de variables) ;
    Résoudre le PLNE par B&B pendant T-MAX secondes ;
    Mettre à jour la solution courante et K et T-MAX ;

Cette méthode a été + ou - formalisé et testé par plusieurs personnes (Prof. Michelon d'Avignon, Ingénieurs ILOG en PPC, …). Michelon, Artigues et Palpant ont des résultats intéressants sur des problèmes d'ordonnancement et d'allocation de fréquences.

Cette approche fonctionne si l'on prend garde à deux points :

- K petit (entre 10 et 50 véhicules) et T-MAX (qq. secondes) ; bien évidemment, il ne faut pas choisir K trop grand, sinon on risque de n'avoir aucune solution au PLNE en T-MAX ; à régler en fonction de la machine et du nombre d'options.

- Remarque expérimentale : ne pas sélectionner les véhicules mobiles par zone contiguë, mais au contraire, sélectionner des positions espacées (les tirer au hasard fait donc l'affaire). Nous avons remarqué que dans ce cas, la relaxation du PLNE était bonne et permettait au B&B de trouver rapidement la solution optimale du PLNE.

Résultat : nous parvenons à explorer de grands voisinages basés sur des K-permutation avec 10 <= K <= 50. Cette approche fonctionne mais converge très lentement. Résolution âr PLNE très lourde (utilisation d'une librairie externe GLPK pourtant efficace).

Relative déception => grands voisinages pas aussi puissants que leur nom le laisse entendre

Une question reste en suspens : pourquoi est-ce que la relaxation est-elle bonne lorsque les véhicules sont suffisamment écartés ?

Nous avons aujourd'hui la réponse. Prenons Q-MAX le plus grand dénominateur parmi toutes les options. Choisissons les K positions mobiles tel qu'écartées d'au moins Q-MAX. Le nombre de violations causé par l'affectation d'un véhicule à une position ne dépend alors de l'affectation d'aucun autre véhicule. => montrer un exemple au tableau.

Ainsi, pour chaque véhicules mobiles et positions mobiles, on peut calculer le coût de l'affectation. Trouver la meilleure affectation devient alors un problème de couplage de coût minimum : « assignement problem ». Peut être résolu par un algorithme combinatoire en O(NMOB^3) avec NMOB nombre de véhicules mobiles.

Nous venons donc de définir un grand voisinage explorable en temps polynomial par couplage. De plus, cela explique pourquoi le PLNE se résout bien dans ce cas : en fait, celui-ci correspond à une formulation PLNE du problème de couplage, or nous savons que la matrice d'un tel programme possède la très bonne propriété d'être unimodulaire : toutes les solutions obtenues par le simplex sont entières. Par conséquent, pas de besoin de B&B, la solution de la première relaxation est la solution optimale entière.

Nous avons utilisé ce voisinage de façon spécifique dans l'heuristique précédente, avec résolution par un algorithme combinatoire très efficace en pratique (meilleur que du N^3 : NMOB * NCL en moyenne). Cela améliore nettement les résultats de l'approche par grand voisinage, mais la convergence rapide bloque à certain niveau : le voisinage ainsi défini, bien que très grand, ne permet pas d'atteindre toutes les solutions (comme le SWAP seul).

Nous avons alors décidé d'hybrider petits et grands voisinages. Nous avons ajouté dans le petit voisinage 0.2 % de grand voisinage par couplage. C'est une réussite, même si les résultats ne sont que légèrement meilleurs : en temps 20 % de gain et en nombre d'itérations 70 %. L'ajout de ce voisinage même en très faible proportion a des effets notables sur la recherche : diversification + grande, convergence + rapide. En particulier, VLNS à un taux de réussite et même de progression très élevé par rapport aux autres transformations (cf. Fig. 9 papier RAIRO).

Par exemple sur l'instance 300-05 très difficile (en 10 mn) :

VFLS	: 27.81,	235 sec,	206 millions d'itérations
HYBRID	: 27.79,	211 sec,	71 millions d'itérations

⇨ Montrer l'idée de l'effet de la VLNS sur la descente au tableau par rapport à la VFLS

Nous n'avons pas fini de travailler sur ce sujet… Beaucoup de questions restent ouvertes !

e) Le problème chez RENAULT

Problème bien réel : ordonnancement dans les usines RENAULT.

Prise en compte des contraintes et objectifs de l'atelier assemblage, mais aussi de l'atelier peinture en amont.

Assemblage : contraintes de ratio pour chaque option (très variables selon les instances, bcp d'options, avec un Q_i très grand, …) ; option classée en 2 catégories : prioritaires (P), non prioritaires (NP) ; objectif : minimiser le nombre de violations sur chaque ratio.

Peinture : chaque véhicule a une couleur en plus des options ; objectif : minimiser le nombre de purges (C) à effectuer, une purge à chaque changement de couleur dans la séquence ; contrainte : le nombre de véhicules consécutifs ne doit pas dépasser un certain nombre PAINT-LIMIT (vérification visuelle de la qualité de la peinture).

Fonction objectif / coût d'une séquence : P << NP << C (P toujours avant NP)
Différents objectifs possibles : P, NP, C
P, C, NP
C, P, NP
P, C
C, P

Problème très difficile. RENAULT avait travaillé sur celui-ci : approche par LS avec recuit simulé (essentiellement des swaps).

On applique la recette :

i) l'heuristique (« le pilote »)
ii) les voisinages (« le volant »)
iii) l'algorithmique de l'exploration (« le moteur »)

- l'heuristique :

Reste simple dans l'esprit : toujours descente avec first-improvement
Fonction de l'ordre des objectifs : OptA, OptB, OptC (cf. Fig.3 papier EJOR)

Remarque : optimiser les purges comme unique objectif est polynomial (glouton)

- les voisinages :

Toujours basé sur SWAP, INSERTION, REFLECTION. Ajout du SHUFFLE (mélange aléatoire sur quelques positions contiguës).

Enrichis et adapté par rapport aux voisinages car sequencing académique. En particulier, spécialisé pour les couleurs (décisif !). Fonction de l'ordre des objectifs : OptA, OptB, OptC (cf. Fig.4 papier EJOR).

- l'algorithmique :

Basé sur ce que nous avons dit pour le car sequencing académique : chaque transformation SWAP, INSERTION, REFLECTION est évalué en O(Q_i) pour chaque option et en O(1) pour détecter une violation de la PAINT-LIMIT et mettre à jour le nombre de purges. Grâce à des structures de données spéciales (mise à jour incrémentale des structures grâce aux invariants).

SHUFFLE coûteux car sans propriétés (à utiliser sur de petites portions de séquence et avec parcimonie). Utile pour diversifier lorsque toutes les autres transformations échouent (optimum local fort).

Schéma d'évaluation spécial (cf. Fig.5 papier EJOR) : PAINT-LIMIT évalué en premier puis les 3 objectifs en cascade. Accélérer encore l'évaluation en classant l'évaluation des options : les options sont triées dans l'ordre décroissant du nombre de violations (les options les plus dures sont évaluées en premier, cf. Fig.6).

⇨ idée : toujours de rejeter une transformation au plus tôt dans le processus d'évaluation.

Résultat : plusieurs millions d'itérations par minute (plus de 20 millions sur certaines instances), soit 200 millions de solutions visitées sur 10 mn. On gagne le challenge catégorie Junior / Senior !

L'algorithme est aujourd'hui en production dans les 17 usines que compte RENAULT dans le monde. Gain important et visible, en particulier sur les purges, car chiffré : chaque purge utilise du solvant pour nettoyer le pistolet, moins de purges = moins de solvant, sur une année de production l'économie est notable !

5) Planification des interventions et des techniciens chez France Telecom

a) Définition du problème

Un ensemble d'interventions et de techniciens sont à planifier sur des journées.

Données : interventions, techniciens, domaines

Les domaines : ce sont les domaines de compétence

Les techniciens : jours d'indisponibilité ; niveau de compétence dans chacun des domaines (niveau 0 : aucune compétence, niveau 4 : expert)

Les interventions :

- précédences (certaines doivent être effectuées avant d'autres) mais pas beaucoup (graphe des précédences peu dense)

- matrice de compétences : pour chaque domaine et chaque niveau, le nombre de technicien requis pour effectuer l'intervention (aspect hiérarchique : si une intervention requiert un technicien de niveau 2 dans un domaine, un technicien de niveau 3 ou 4 fait l'affaire)

- une priorité : 1, 2, 3, 4 (priorité 1 = urgent)

Objectif : Planifier chaque jour les techniciens en équipe (ils restent ensemble toute la journée) et leur affecter des interventions avec 2 contraintes dures : les compétences et les précédences.

La fonction objectif est : minimiser la date de fin des interventions de priorité 1, puis la date de fin des interventions de priorité 2, puis 3, puis 4.

Problème très difficile, combinatoire explosive ! (plus dur que le problème RENAULT ?)

Comment s'y prendre ? La recette évidemment !

b) Heuristique générale
c) Recherche locale
d) Résultats

Cf. papier ROADEF 2007.

6) Conclusion : les ingrédients pour une recherche locale efficace

Encore et toujours : la recette : (i) l'heuristique, (ii) les voisinages, (iii) l'algorithmique

- a) Diversifier les voisinages / la recherche
- b) Accélérer l'exploration / évaluation des voisinages
- c) La clé de la réussite : le « volume » de transformations

Inviter les élèves à aller regarder les papiers RAIRO, EJOR, ROADEF, JFPC sur le site :

http://www.lif-sud.univ-mrs.fr/~gardi/

Voir également le site de M. Helsgaun pour récupérer papier sur TSP (attention : 71 pages !)

Tout cela peut être retrouvé par recherche sur Google !